CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

#61 AUGUST 1995

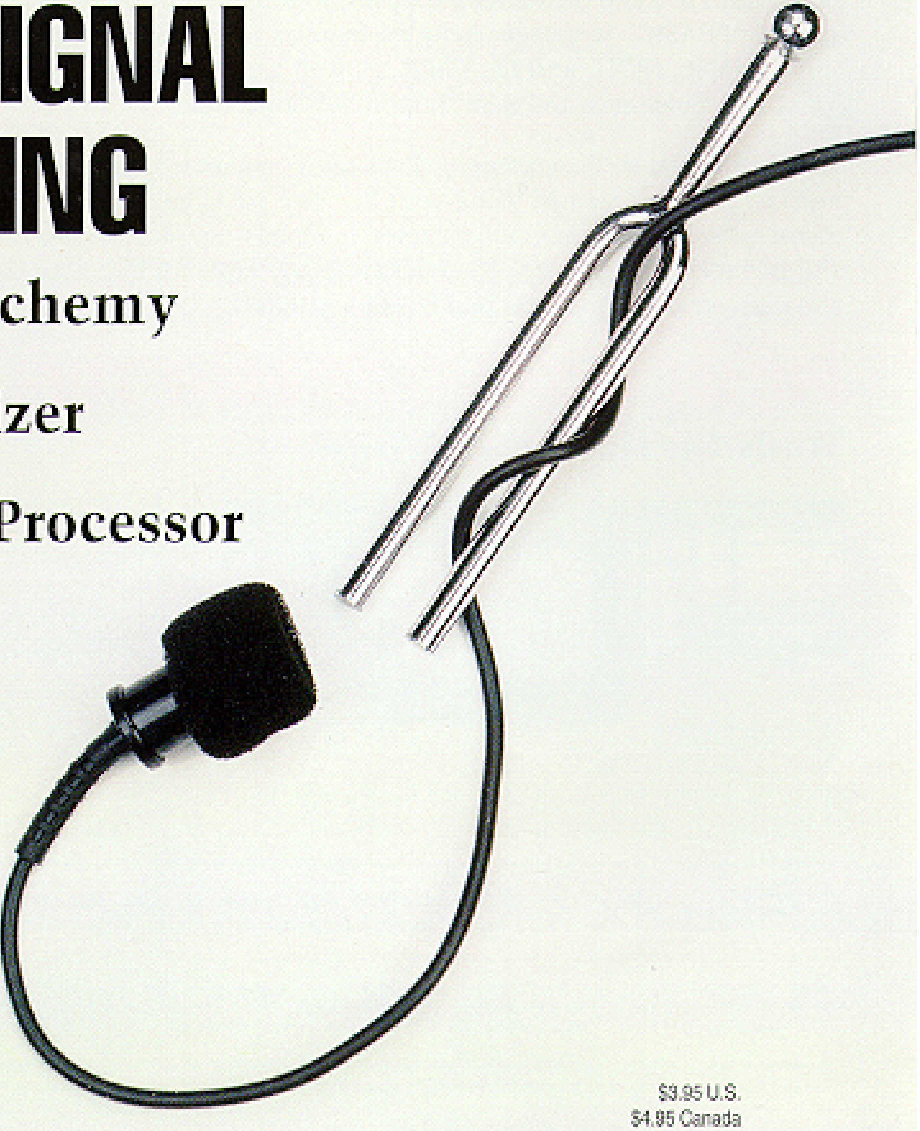
DIGITAL SIGNAL PROCESSING

Digital Filter Alchemy

PC-based Equalizer

Digital's Alpha Processor

Exploring EMI Testing



$3.95 U.S.
$4.95 Canada

EDITOR'S INK

Digital Sound Processing?

have you ever noticed how the "signal" in so many digital-signal processing applications is actually sound? Phone companies constantly try to squeeze more conversations through the same-size pipe without sacrificing sound quality. Consumer electronics boxes try to make tiny headphones sound like studio-quality speakers or your living room like Carnegie Hall. Your office Mac or PC sounds more like a Three Stooges movie than a professional environment. Admit it: you're sick of that plain old error beep and have gone to the other extreme with goofy error sounds.

This month's issue on DSP has no shortage of sound-related signal-processing articles. In our first feature, we skip the tutorial on what digital filters are and move ahead to some techniques for implementing them. Do-While Jones concentrates on design criteria and lets you worry about the platform.

Next, when none of the popular voice-compression schemes would work for his application's design goals, Sébastien Roy came up with his own method. It achieves 2:1 compression, allows the use of a relatively slow and very small processor, and doesn't sacrifice speech quality. After an excellent overview of commonly used techniques, he describes his solution.

Several issues ago, we had an article on using some of the newer DSP-based PC sound cards for doing general-purpose signal processing. Matt Park extends that idea in our next article to include almost any PC sound card on the market.

Speaking of PC sound cards, have you ever wanted to fine tune the frequency content of the final sound in a manner similar to the equalizer on your home stereo? Eric Ambrosino has put together a plug-in board that provides such equalizer functionality for audio produced on your own PC. On-screen displays let you adjust the sound from within Windows.

Finally, we feature Alpha 21164, Digital's latest screamer. Clock speeds continue to rise and cycle times are entering the realm of TTL gate delays.

The issue wraps up with our columns. Ed finishes his look at the PC keyboard, Jeff checks out some inexpensive tools for doing your own preliminary EMI testing, Tom tries to make sense of the latest crop of PLDs, and John explores a new processor from Dallas Semiconductor that transforms the art of power management into a science.

Ken

CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

FOUNDER/EDITORIAL DIRECTOR
Steve Ciarcia

EDITOR-IN-CHIEF
Ken Davidson

TECHNICAL EDITOR
Janice Marinelli

EDITORIAL ASSISTANT
Beth Andrix

ENGINEERING STAFF
Jeff Bachiochi & Ed Nisley

WEST COAST EDITOR
Tom Cantrell

CONTRIBUTING EDITOR
John Dybowski

NEW PRODUCTS EDITOR
Harv Weiner

ART DIRECTOR
Lisa Ferry

PRODUCTION STAFF
John Gorsky
James Soussounis

CONTRIBUTORS:
Jon Elson
Tim McDonough
Frank Kuechmann
Pellervo Kaskinen

PUBLISHER
Daniel Rodrigues

PUBLISHER'S ASSISTANT
Sue Hodge

CIRCULATION MANAGER
Rose Mansella

CIRCULATION ASSISTANT
Barbara Maleski

CIRCULATION CONSULTANT
Gregory Spitzfaden

BUSINESS MANAGER
Jeannette Walters

ADVERTISING COORDINATOR
Dan Gorsky

CIRCUIT CELLAR INK, THE COMPUTER APPLICATIONS JOURNAL (ISSN 0896-8985) is published monthly by Circuit Cellar Incorporated. 4 Park Street, Suite 20, Vernon, CT 06066 (203) 875-2751. Second class postage paid at Vernon, CT and additional offices. One-year (12 issues) subscription rate U.S.A. and possessions $21.95, Canada/Mexico $31.95, all other countries $49.95. All subscription orders payable in US. funds only, via international postal money order or check drawn on US. bank. Direct subscription orders and subscription related questions to Circuit Cellar INK Subscriptions, P.O. Box 698, Holmes, PA 19043.9613 or call (800) 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

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) 985-8457

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, 8 bits, no parity, 1 stop bit, (203) 871-1988; 24001 9600 bps Courier HST. (203) 871.0549

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

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

INSIDE ISSUE 61

READER'S INK

EMBEDDED BENCHMARKS-NOT UP TO SNUFF

Rick Naro's article, "Characterizing Processor Performance" (INK 57), prompts me to write lamenting the dismal lack of embedded processor benchmarks.

The PC and workstation crowd have their SI index and SPECmarks. Sure, these metrics have weaknesses and loopholes, but at least they maintain a semblance of impartiality and run on real hardware.

Embedded processor manufacturers, on the other hand, blithely disclaim that "the only good benchmark is your actual application." They then immediately tout their chips' advantages with fractional precision.

These days, the benchmark of marketing choice is the Dhrystone, a synthetic "toy" program that little reflects the realities of embedded designs, esecially when it comes to such things as interrupts and I/O.

Breathless Dhrystone claims are rarely accompanied by any description of hardware. That's understandable since it's quite likely the chip and/or board design were merely simulated. Unfortunately, simulated 5-ns SRAMs are a lot cheaper than the real thing.

The same goes for the C compiler-the R&D model with hardwired benchmark optimizations. Perhaps, the optimization qualifiers, initially just buried in the benchmark report, get relegated to the status of distracting details, then fine print, and then nonexistent print.

I sure wish the researchers who dissect workstation CPUs down to the last nanosecond would shift some of their attention to the embedded world. Maybe the diversity of embedded applications means generic, yet realistic, benchmarking isn't possible. If that's the case, so be it.

Regardless of whether we get some good benchmarks or just throw up our hands and quit, couldn't we at least dispatch with all the Dhrystone hype?

Bruce **Reinhart**
Baltimore, MD

GRAPHICALLY SPEAKING

I am impressed with the breadth and scope of the articles in your graphics issue (INK 60).

Mike Bailey's "Using Color In Scientific Visualization" is one of the best introductions to color and human perception I have read. It is too bad that these principles were not used in the ad on page 26 of the same issue.

In a former job, I used a Polhemus Isotrak as a 6°-of-freedom mouse for scientific visualization. I wish the Murry and Schneider article had been available when I was trying to maximize the performance of my Isotrak.

The third article by James Goel accurately depicts the problems in resizing images and video compression. However, as desktop computers become faster and parallel computing becomes norm, specialized hardware for resizing and compression will not be necessary.

Juseppi **Caboto**
Chicago, IL

THE CORRECT CAN LABEL

The Controller Area Network (CAN) articles published in *INK 58* and 59 reference a Philips PCA-80C200. The actual Philips part number is PCA82C200. The part is referenced properly in the schematics, but all references in the text to a 80C200 should read 82C200.

Brad Hunting
Cohoes, NY

CATCH THE SHUTTLE

On page 12 of *INK 58,* there is a "New Product News" listing for Shuttle Technology, Inc. The correct address for that company is 43218 Christy St., Fremont, CA 94538. Send E-mail to sales@shuttletech.com.

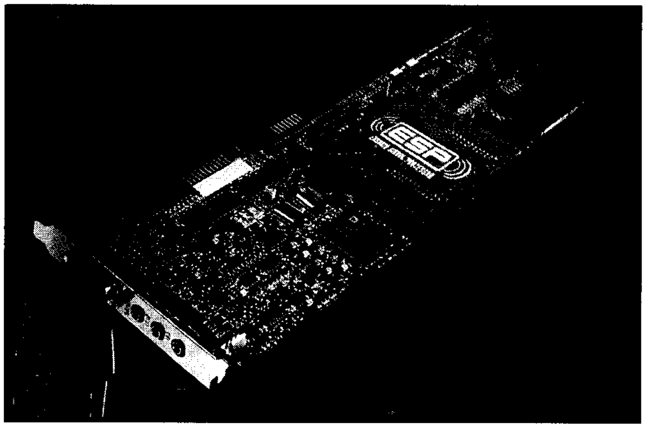
Contacting Circuit Cellar

We at the *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 off ices at (203) 875-2199.
Fax: All faxes may be sent to (203) 872-2204.
BBS: All of our editors and regular authors frequent the Circuit Cellar BBS and are available to answer questions. Call (203) 871-1988 with your modem (300–14.4k bps, 8N1).
Internet: Electronic mail may also be sent to our editors and regular authors via the Internet. To determine a particular person's Internet address, use their name as it appears in the masthead or by-line, insert a period between their first and last names, and append "@circellar.com" to the end. For example, to send Internet E-mail to Jeff Bachiochi, address it to jeff.bachiochi@circellar.com. For more information, send E-mail to info@circellar.com.

NEW PRODUCT NEWS

Edited by Harv Weiner



DSP-BASED AUDIO BOARD

The Soundscape Elite from Ensoniq is a PC sound board that features custom dual DSPs to provide 16-bit, 32-voice wavetable sound along with multiple, simultaneous, downloadable, real-time effects. The Soundscape Elite board incorporates the same custom wavetable synthesizer (OTTO) and digital signal effects processor used in the company's keyboards to create rich, realistic sound and versatile effects. The Elite runs special effects like reverb, chorus, flange, distortion, pitch and phase shift, as well as equalization in simultaneous real time. These effects add depth and clarity to the sound. The board can also be upgraded with other special effects available from the manufacturer.

Soundscape Elite contains a ROM/RAM-based wavetable sound set that is Extended MIDI compatible with 128 instruments and 61 drums, the full GS set with 7 drum kits, and the MT-32 instrument set. It supports four CD-ROM interfaces including IDE, Sony, Panasonic, and Mitsumi as well as Windows 95, NT, and OS/2 operating systems.

Soundscape Elite has a suggested list price of $289.

Ensoniq, Inc.
155 Great Valley Pkwy.
Malvern, PA 19355
(610) 647-3930
Fax: (610) 647-8908

#500

DSP BOARD WITH ANALOG AND DIGITAL I/O

Dalanco Spry announces the Model 5000 **Digital Signal Processing Board** with analog and digital I/O. The unit is designed for IBM PC/AT and ISA bus-compatible microcomputers. Applications include data acquisition, instrumentation and control, speech and audio, as well as general-purpose DSP software development.

Model 5000 is based on the TI TMS320C5 l-80 40-MIPS DSP and provides data acquisition for 8 channels at 12-bit resolution and a maximum 500-kHz sampling rate. Two 12-bit analog output channels, a buffered digital I/O connector for user expansion, and a serial interface to the TI DSP are included.

The board is populated with 64K words of zero-wait-state program RAM and 128K words of dual-ported data RAM. Program memory may be accessed at all times by the host for code downloading and debugging. The dual-ported architecture enables users to create applications requiring simultaneous mathematical calculations and analog I/O coupled with concurrent disk I/O.

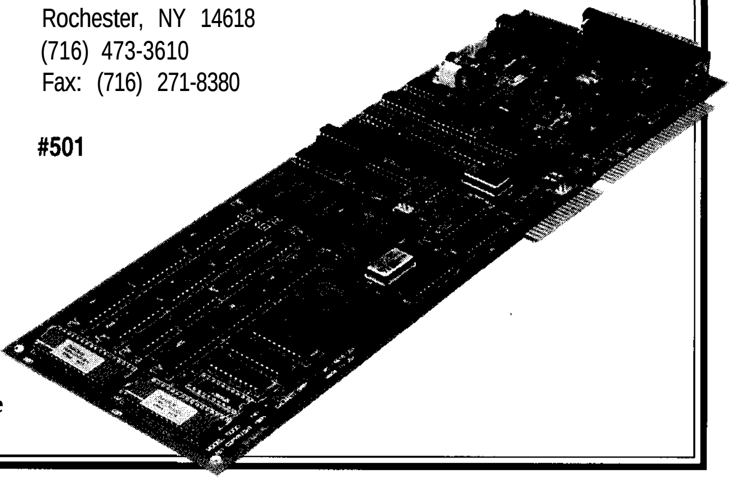
The high throughput of the Model 5000 to the host PC's memory (up to 3 MBps) and disk make it ideal for data logging and transfer tasks. Multiple boards may be accommodated within a single system.

All required development tools are included with the Model 5000. The A5000 assembler loads executable code directly into the Model 5000 after successful assembly. Optionally, it produces an ASCII list file of the executable for inclusion in the user's host-based programs. The D5000 debugger includes inline assembler and disassembler, breakpoint, trace, single step, and modification and viewing of memory and registers. A link package features board-management functions in source code, which links to most high-level languages for both DOS and Windows. Also included is an object file loader for the assembler and C compiler from TI, programming examples with the TI compiler, and Visual Basic for Windows. Application software is also provided.

Model 5000 sells for $1300 and includes application and development software.

Dalanco Spry
89 Westland Ave.
Rochester, NY 14618
(716) 473-3610
Fax: (716) 271-8380

#501



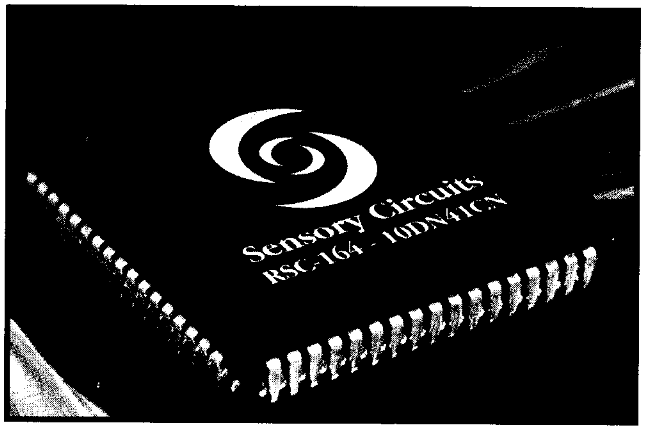
NEW PRODUCT NEWS

LOW-COST SPEECH RECOGNITION

Sensory Circuits introduces a low-cost integrated circuit which includes speech recognition and synthesis, audio record and playback, and a general-purpose 4-MIPS 8-bit microcontroller. The RSC-164 is targeted at cost-sensitive consumer electronics where speech recognition has been cost prohibitive. Typical applications include command and control, interactive devices, and stand-alone interactive training and teaching.

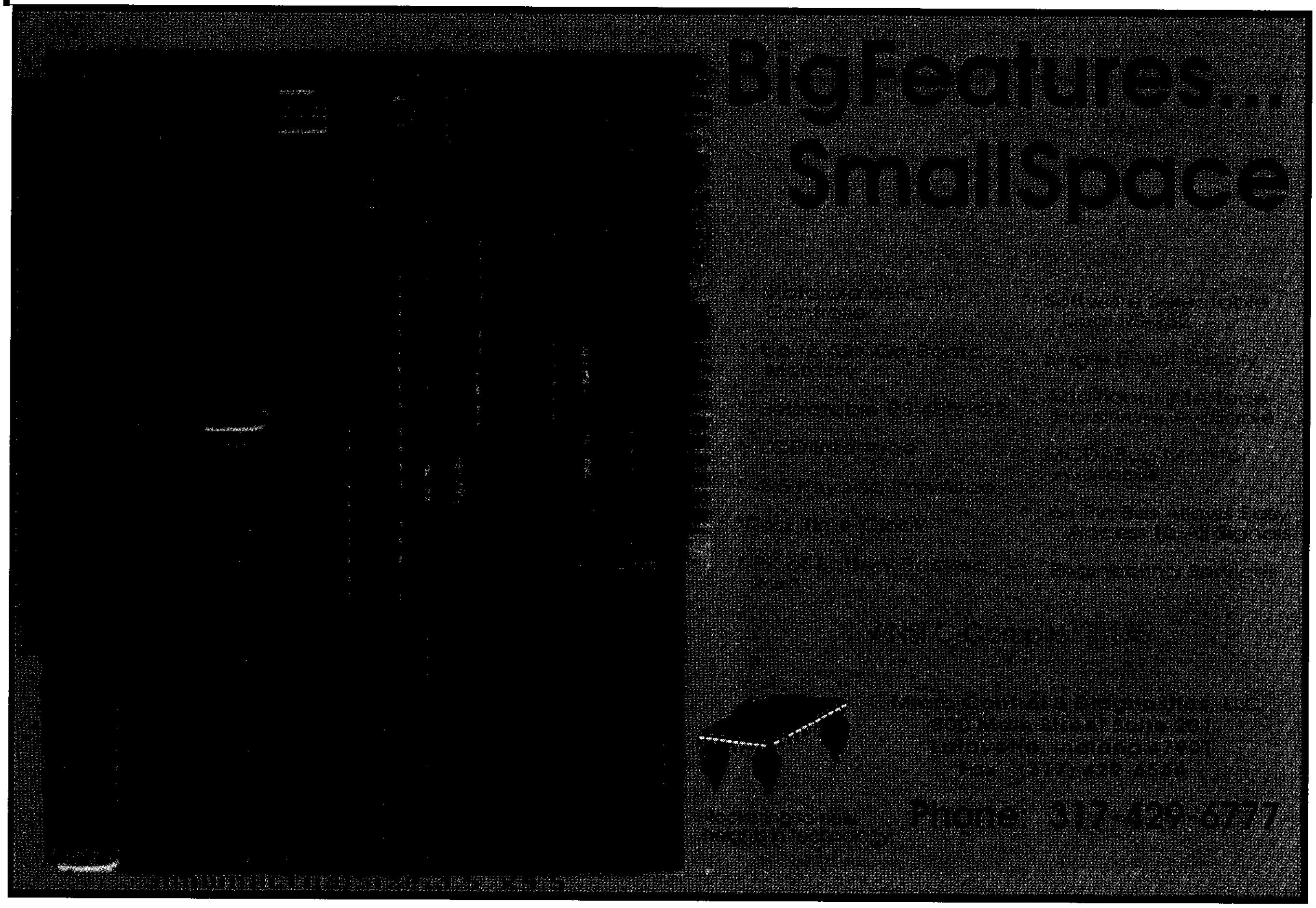
The RSC-164 provides over 40 s of on-chip, high-quality, digitized speech playback or sound effects and recognizes hundreds of words. An on-chip 4-MIPS processor with 64-KB of ROM provides general-purpose product control. The RSC-164 can address off-chip ROM for large-vocabulary applications and can record and play back messages by addressing external memory.

The RSC- 164 uses a neural network-based, speaker-independent recognition technology. User training is unnecessary. It works with a wide range of voice qualities, ages, and accents, achieving 98% recognition accuracy without using DSPs or a RAM-intensive architecture. The chip is a CMOS design with on-chip powerdown to minimize battery drain. Control and communications with off-chip devices are provided through 16 programmable I/O lines. Development tools will be released by the end of 1995 for customer code development.



The RSC- 164 sells for under $4 in large quantities.

Sensory Circuits, Inc.
1735 N. First St., Ste. 313 • San Jose, CA 95112-4511
(408) 452-I 000 • Fax: (408) 452-I 025 **#502**



#104

NEW PRODUCT NEWS

LOW-COST DSP DEVELOPMENT SYSTEM

Analog Devices has a complete DSP development system for $89. The **EZ-KIT Lite** is a low-cost demonstration and evaluation kit for ADSP-2100 family designs. The system includes a DSP development board featuring a 33-MIPS ADSP-2181 DSP with 32-KB words of on-chip memory. The board also contains an AD1847 stereo-audio SoundPort codec, an RS-232 interface based on the ADM232 chip, an analog-input signal-conditioning circuit based on the SSM2135 dual op-amp, and an EPROM socket. The board can be connected to the COM port of a PC and can be configured for stand-alone mode with programs developed by the user.

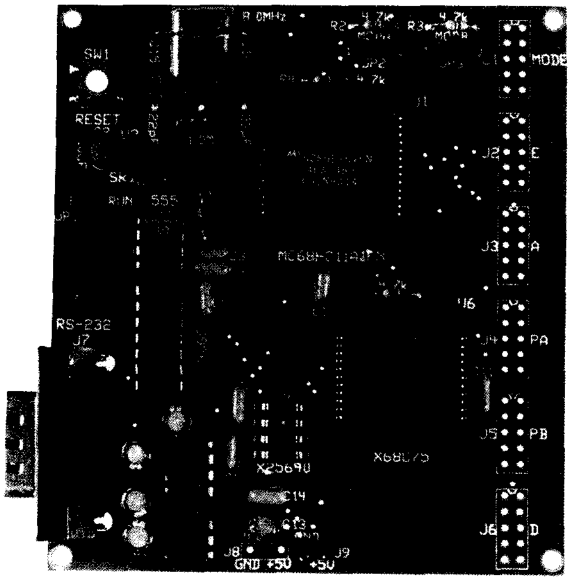
The EZ-KIT Lite includes a monitor program that runs on the DSP and a Windows-based host program that runs on the PC. These programs let the user interactively download code to the onboard DSP. The user can also download and upload information to and from the DSP's internal memory. An assembler, linker, simulator, and PROM formatter are also included.

The EZ-KIT Lite sells for $89 and includes the EZ-LAB board, DSP demo software, host PC software, DSP monitor, development software set, a collection of demos with source code, documentation, an RS-232 cable, and a 9-V power supply.

Analog Devices
One Technology Way • **Norwood,** MA 02062-9106 • (617) **461-3881** • Fax: (617) 461-3010 **#503**

SINGLE-BOARD COMPUTER

LDG Electronics releases a low-cost, single-board computer. The **SBC-8K** microcontroller, intended for embedded control, data logging, and educational applications, is one of the first 68HC1 l-based development systems available with the Xicor X68C75(8-KB EEPROM and port replacement device).



The board comes with a DE-9 RS-232 port, eight 8-bit A/D converters, 16 programmable and 8 fixed I/O lines, 8704 bytes of EEPROM, 272 bytes of RAM, and a socket for an optional 8-KB serial EEPROM. Operation is from a single 5-V source and it draws less than 60 mA.

Through the new **CodeLoad+ 2.0** development software, users can program the internal, external, or serial EEPROM with S19 records via RS-232. The Memory Dump command displays internal, external, or serial memory contents and stores to a PC file. Program code can also be executed via RS-232. The terminal emulator lets users send ASCII codes to the SBC-8K and log RS-232 data to a file.

The software development package included with the SBC-8K contains CodeLoad+ 2.0, source files for all sample programs, and complete documentation. The SBC-8K sells for $79.95 and the 8-KB serial EEPROM option sells for $10.

LDG Electronics
1445 **Parran** Rd.
St. Leonard,. MD 20685
(410) 586-2177

#504

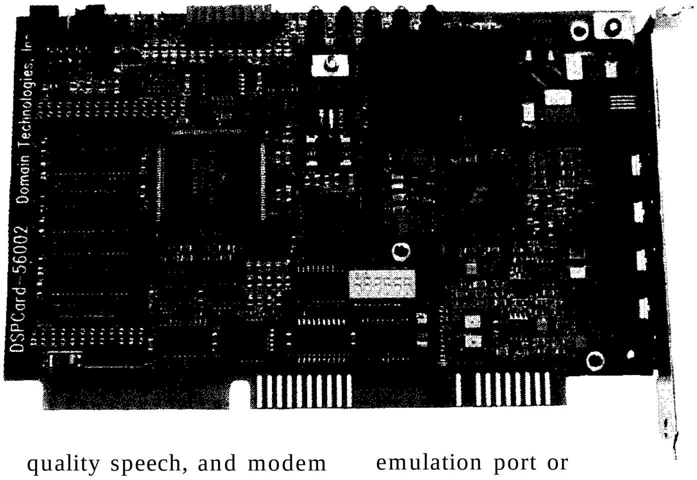
NEW PRODUCT NEWS

DSP SOFTWARE DEVELOPMENT PLATFORM

Domain Technologies is shipping the DSP56002 **Development** System, an integrated software development platform for the Motorola DSP56002. The system provides all the tools required to develop, test, and run DSP56002 software. It includes a 24-bit assembler, a source-level symbolic debugger, and a high-performance DSP card. The toolset runs in a wide range of applications including multimedia, telecommunications, audio signal processing, and data acquisition.

The PC plug-in card consists of a 20-MIPS DSP56002 chip, static memory, stereo 16-bit analog I/O, telephone interface, and two PC interfaces. The card has an open architecture and is supplied with hardware details and software drivers.

The analog I/O unit is a single-chip stereo multimedia codec. It supports CD-quality music, telephone-quality speech, and modem signals. The A/D and D/A converters are 64 times oversampled delta-sigma converters with on-chip filters. A DAA lets the card connect directly to telephone lines. The telephone interface is FCC approved and can be used for voice or data applications up to V.TURBO. The DSP card is populated with 128K words of zero-wait-state RAM. The DSP56002 interfaces with the PC through the OnCE emulation port or through the high-speed host port.



The source-level debugger runs under Windows and supports software development in assembly and C. Through the debugger, the user can load DSP programs, examine memory, set breakpoints, single step, examine data structures, benchmark a DSP program, plot memory graphically, and so on. The debugger interfaces with the DSP through the 56002's dedicated OnCE emulation port.

The 24-bit symbolic assembler fully complies with the Motorola DSP56002 instruction set. The assembler produces a COFF load file with source-line information and an optional listing file.

The DSP56002 Development System sells for $800.

Domain Technologies, Inc.
1700 Alma Dr., Ste. 245
Plano, TX 75075
(214) 985-7593
Fax: (214) 985-8579
domain@metronet.com

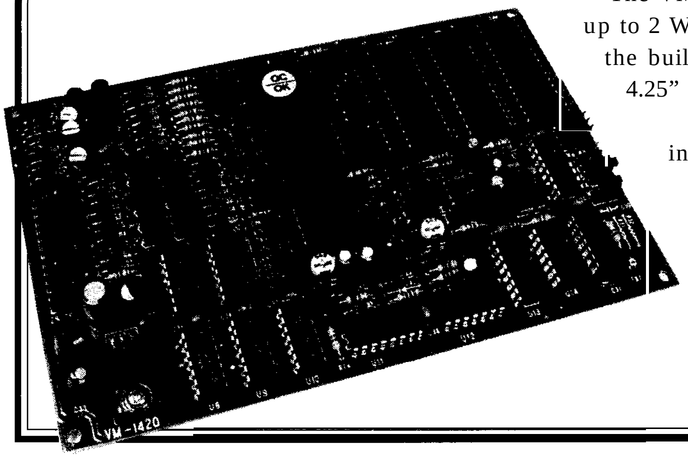
#505

SELF-CONTAINED VOICE BOARD

The **VM1420** Voice-Board Module is now available from Eletech Electronics. Built with advanced digital voice technology, the VM1420 provides low-cost, high-quality audio. The board is totally self-contained, needs no controller to operate, and features 20 trigger input pins. When activated by external contact closures or motion sensors, it plays 1 of the 20 messages stored in its EPROM chip. Up to 17 minutes of messages can be custom programmed into EPROMs with a low-cost voice-development tool.

The VM1420 runs off a single 6-12-V DC supply. Audio output is up to 2 W into a 4-Q speaker. Standby current is about 50 µA when the built-in power management circuitry is enabled. Measuring 4.25" x 6.5", the board easily fits into tight enclosures.

The VM1420 offers audio output in talking displays, industrial control, talking alarms, and so on. The VM1420 sells for $80 in quantity of 10 (without EPROM].



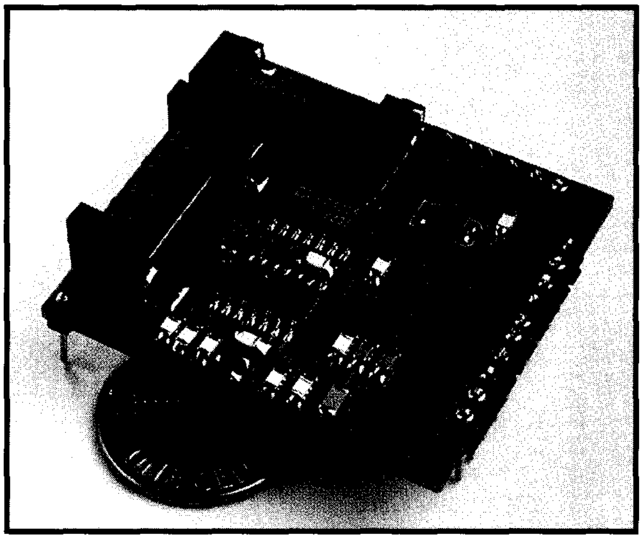
Eletech Electronics, Inc.
16019 Kaplan Ave. • Industry, CA 91744
(818) 333-6394 • Fax: (818) 333-6494

#506

NEW PRODUCT NEWS

ELECTRONIC COMPASS MODULE

Traditional electronic compass modules cost $500–$1000 and are so large that integrating them into systems such as remote-operated vehicles and hand-held GPS receivers is difficult and time consuming. Precision Navigation introduces an alternative that costs $50 and has a footprint smaller than a matchbox.



The **Vector-2X** is a 2-axis strapped-down magnetometer (i.e., no moving parts) that uses proprietary technology. As a magneto-inductive technology, this sensor consumes less than one third the power of alternative fluxgate compasses and is accurate to within 2” RMS. More advanced modules offer tilt compensation or accuracy under 1” RMS. These advanced sensors maintain accuracy in highly dynamic applications such as aviation and navigation.

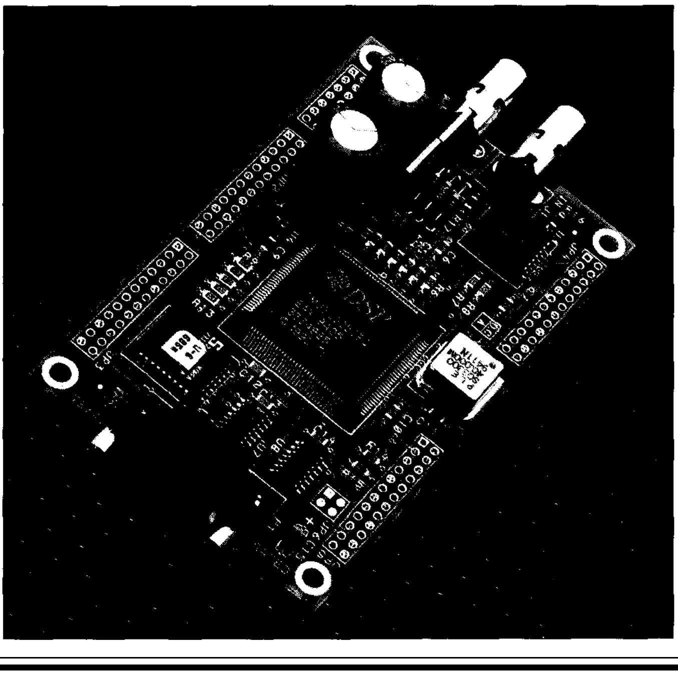
The Vector-2X can be used in a multitude of applications including auto and industrial navigation systems, robotics, consumer and hobbyist markets, as well as hand-held GPS receiver and survey instruments. The unit features hard-iron calibration and a 10-Hz sampling rate (in continuous mode). It operates on a single 5-V power supply, drawing less than 10 mA (it uses less than 1 mA in power-down mode). The unit also features a 3-wire serial output format (compatible with Motorola SPI and National Microwire) and a pin-selectable BCD or binary output format. Master or slave capability for data-clock generation is provided, and continuous or polled modes are available.

Precision Navigation, Inc.
1235 Pear Ave., Ste. 111 • Mountain View, CA 94043 • (415) 962-8777 • Fax: (415) 962-8776

#507

DSP-BASED FUZZY-LOGIC SYSTEM

Texas Instruments releases development software for fuzzy-logic systems based on the TMS320 DSP family. ***fuzzy*****TECH MCU-320 Explorer V4.0** combines the fuzzy system development and code-generation features of the MCU-320 software tools with the assembly, debug, and real-time execution functions of the **TMS320C50 DSP Starter Kit.** Its Windows-based development environment contains the tools needed for design, optimization, and verification. An animated simulation of a crane controller enables users to experiment with fuzzy rules and system structure.



Version 4.0 enhancements include a 3D analyzer with real-time tracing for rule base verification, improved variable editors, statistical-debug and serial-debug modes to support rule-based optimization, and remote debugging over the serial port. Also included are M code generation for Matlab or Simulink, DDL or DDE support to integrate *fuzzy*TECH with other design software, and a Fuzzy Design Wizard for automated prototype generation.

The Starter Kit sells for $99 and the *fuzzy*TECH MCU320 Explorer for $199.

Texas Instruments
P.O. Box 172228
Denver, CO 80217-9271
(800) 477-8924, Ext. 4006
(303) 294-3747, Ext. 4006

#508

FEATURES

FEATURE ARTICLE

Do-While Jones

Digital Filter Alchemy

Turning Circuits into Code

Do-While assumes you've picked your microprocessor and hardware circuitry. He just wants to help you with the digital-filtering techniques. He offers closeups on actual digital-filter implementation.

Ohis article isn't about digital filter design-it's about digital filter implementation, and there is a difference. Filter design involves the process of figuring out the characteristics of the filter you need. In this article, I assume:

- you know what kind of filter you need
- you know how to build an analog circuit and its desired characteristics
- you want help implementing that filter digitally

This article gives you the process for transmuting analog circuits into computer code.

Let's start with a hands-on look at two simple filters before worrying about the general solution. Consider the single-pole low-pass and single-pole high-pass filters shown in the first two rows of Figure 1. If you compare the two circuit diagrams from Kirkoff's point of view, they are identical.

To analyze either circuit, you must compute the current flowing through a resistor and capacitor in series. Knowing the current, you can determine the voltage across the capacitor. The only difference is how you define the output. In the low-pass case, the output is the voltage across the capacitor while in the high-pass

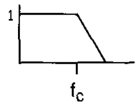
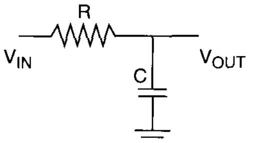
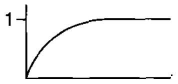
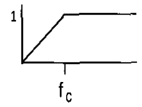
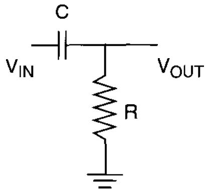
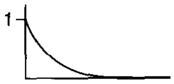
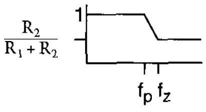
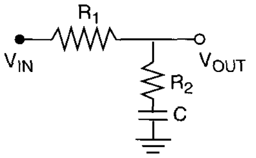
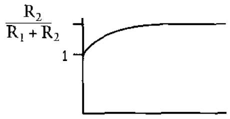
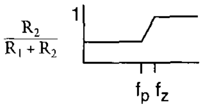
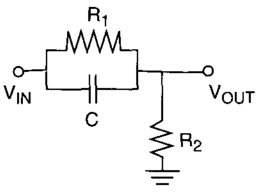
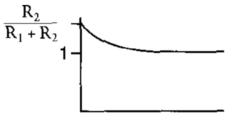
Filter Type	Circuit Model	Transfer Function (frequency domain)	Unit Step Response (time domain)
Single-Pole Low Pass  	 	$\frac{\omega_c}{s+\omega_c}$ $\omega_c = \frac{1}{RC}$ $\omega_c = 2\pi f_c$	 $y(t) = 1 - e^{-\omega_c t}$
Single-Pole High Pass  	 	$\frac{s}{s+\omega_c}$ $\omega_c = \frac{1}{RC}$ $\omega_c = 2\pi f_c$	 $y(t) = - e^{-\omega_c t}$
Single-Pole Lag/Lead  	 	$\frac{\omega_p(s+\omega_z)}{\omega_z(s+\omega_p)}$ $\omega_z > \omega_p$ $\omega_z = \frac{1}{R_2C}$ $\omega_p = \frac{1}{(R_1+R_2)C}$	 $y(t) = 1 - \frac{\omega_z - \omega_p}{\omega_z} e^{-\omega_p t}$
Single-Pole Lead/Lag  	 	$\frac{s + \omega_z}{s+\omega_p}$ $\omega_p > \omega_z$ $\omega_z = \frac{1}{R_1C}$ $\omega_p = \frac{R_1+R_2}{R_1R_2C}$	 $y(t) = \frac{\omega_z}{\omega_p} + \left(1 - \frac{\omega_z}{\omega_p}\right) e^{-\omega_p t}$

Figure la-Common filters can be expressed by their frequency response, circuit model, *Laplace* transform, or *unit* step response.

case, the output is the input voltage less the voltage across the capacitor. So, if you can compute the voltage across the cap, you can determine the output voltage for either filter.

Recall the step response for filters. The low-pass filter is:

$$y(t) = 1 - e^{-2\pi F_c t}$$

and the high pass is:

$$y(t) = e^{-2\pi F_c t}$$

where F_c is the cutoff frequency in hertz. Often, it is more convenient to specify ω_c as the cutoff frequency in radians per second, which is simply 2π x F_c. This substitution simplifies the equations slightly by eliminating the need to explicitly write out the 2π term. Since ω_c equals $2\pi F_c$, the equations become:

$$y(t) = 1 - e^{-\omega_c t}$$

for the low pass and

$$y(t) = - e^{-\omega_c t}$$

for the high pass.

The corresponding digital filters must have the same step responses. 1’ll use this fact later in the implementation of the digital filters, thereby completely avoiding the need for Z-transforms.

Figure 2 shows the common form for a single-pole infinite-impulse-response digital filter. The box marked Z^{-1} represents a one-sample delay. That is, node 1 contains the value node 0 had at the last sample time. Node 0 is the sum of the input value, X, plus the product of *Al* and the value of node 1. The output, Y, is the sum of *BO* times the value of node 0 plus *B1* times the value of node 1. This filter is easily implemented on a computer with three statements (see Listing la).

Notably, the sequence of the equations is important. You must compute them in the order given because each equation depends on intermediate calculations from the previous equation.

Let’s reverse-engineer this filter and see what it does. Consider its response to a unit step input. For all times T < 0, everything must be initialized to 0, so Y = 0 (see Listings lb and c).

For large values of T, when the output has reached a steady state, it must be the case that N 0 DE_1 = N 0 DE_0 because nothing is changing. The input X is still 1 because it is a unit step (see Listing 1d}.

Let’s pick some values and see what happens when *AZ = 0.5, BO = 0,* and *B1 = 0.5.* The equation in Listing lc shows that the initial response to

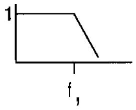
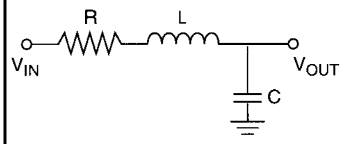
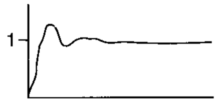
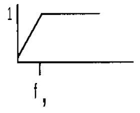
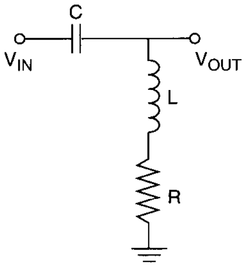
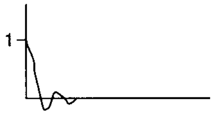
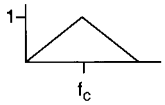
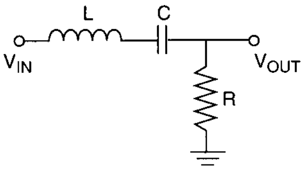
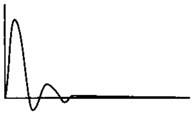
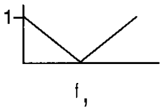
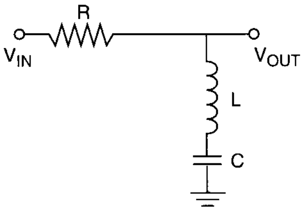
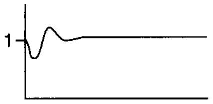
Filter Type	Circuit Model	Transfer Function (frequency domain)	Unit Step Response (time domain)
Double-Pole Low Pass  	 	$\frac{\omega_n^2}{s^2+\zeta\omega_n s+\omega_n^2}$ $\omega_n = 2\pi f_c$ $\omega_n = \sqrt{\frac{1}{LC}}$ $\zeta = \frac{R}{2}\sqrt{\frac{C}{L}}$	 $y(t) = 1 - \frac{\omega_n}{\omega_d} e^{-\alpha t} \sin(\omega_d t + \phi$ $\omega_d = \omega_n\sqrt{1-\zeta^2}$ $\alpha = \zeta\,\omega_n$
Double-Pole High Pass  	 	$\frac{s(s+2\zeta)}{s^2+2\zeta\omega_n s+\omega_n^2}$ $\omega_n = 2\pi f_c$ $\omega_n = \sqrt{\frac{1}{LC}}$ $\zeta = \frac{R}{2}\sqrt{\frac{C}{L}}$	 $y(t) = \frac{\omega_n}{\omega_d} e^{-\alpha t} \sin(\omega_d t + \phi)$ $\omega_d = \omega_n\sqrt{1-\zeta^2}$ $\alpha = \zeta\,\omega_n$ $\phi = \tan^{-1}\left(\frac{\omega_d}{\alpha}\right)$
Double-Pole Band Pass  	 	$\frac{2\zeta\omega_n s}{s^2+2\zeta\omega_n+\omega_n^2}$ $\omega_n = 2\pi f_c$ $\omega_n = \sqrt{\frac{1}{LC}}$ $\zeta = \frac{R}{2}\sqrt{\frac{C}{L}}$	 $y(t) = \frac{2\alpha}{\omega_d} e^{-\alpha t} \sin(\omega_d t)$ $\omega_d = \omega_n\sqrt{1-\zeta^2}$ $\alpha = \zeta\,\omega_n$
Double-Pole Notch Pass (Band Stop)  	 	$\frac{s^2+\omega_n^2}{s^2+2\zeta\omega_n+\omega_n^2}$ $\omega_n = 2\pi f_c$ $\omega_n = \sqrt{\frac{1}{LC}}$ $\zeta = \frac{R}{2}\sqrt{\frac{C}{L}}$	 $y(t) = 1 - \frac{2\alpha}{\omega_d} e^{-\alpha t} \sin(\omega_d t)$ $\omega_d = \omega_n\sqrt{1-\zeta^2}$ $\alpha = \zeta\,\omega_n$

Figure la-continued

the step is 0. Listing 1d, however, says the final value is 1 .0.

Compute the statements in Listing la and you get the values in Table 1. If you plot the values, it looks mighty familiar:

$$y(t) = 1 - e^{-\omega_c t}$$

But, is this plot an exact exponential curve or just an approximation? What is the value of ω_c?

Let's assume that it is exponential and try to find the value of ω_c. For convenience, assume that t is measured in seconds and there is one sample per second. We need a value for ω_c such that:

$$y(t) = 1 - e^{-\omega_c \times 0}$$
$$= 0$$

Since any positive value short of infinity works, this isn't much help. Instead, we need to seek the value of ω_c such that:

$$y(t) = 1 - e^{-\omega_c \times 1}$$
$$= 0.50000$$

The only value of ω_c that works is 0.6931471. If you look for a value of ω_c such that:

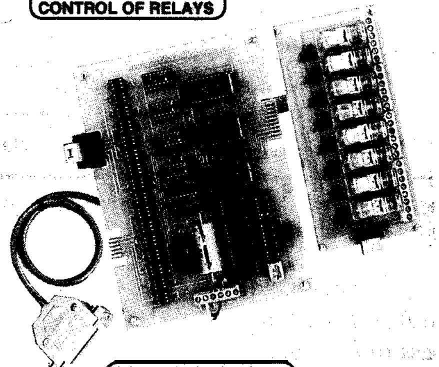
$$y(t) = 1 - e^{-\omega_c \times 2}$$
$$= 0.75000$$

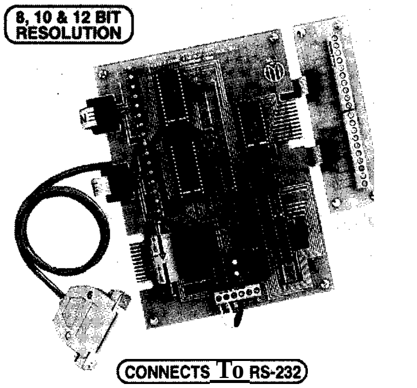
you'll find that the same value (i.e., 0.6931471) solves that equation too. In fact, 0.6931471 works for all the other transition values as well. So, I can use statements in Listing la to simulate the response of a simple RC low-pass filter if I let *B0* = 0 and pick *Al* and *B1* properly.

Filter Type	Digital Filter Coefficients
Single-Pole Low Pass	$A1 = e^{-\omega_c / f_s}$ $BO=O$ $B1 = 1 - A1$
Single-Pole High Pass	$A1 = e^{-\omega_c / f_s}$ $B0 = 1$ $B1 = -1$
Single-Pole Lag/Lead	$A1 = e^{-\omega_p / f_s}$ $B0 = \frac{\omega_p}{\omega_z}$ $B1 = 1 - A1 - B0$
Single-Pole Lead/Lag	$A1 = e^{-\omega_p / f_s}$ $B0 = 1$ $B0 = \frac{\omega_z}{\omega_p}(1 - A1) - 1$
Double-Pole Low Pass	$A1 = \frac{y\left(\frac{3}{f_s}\right) - y\left(\frac{1}{f_s}\right) - y\left(\frac{2}{f_s}\right) + y\left(\frac{1}{f_s}\right)y\left(\frac{2}{f_s}\right)}{y\left(\frac{2}{f_s}\right) - 2y\left(\frac{1}{f_s}\right) + y\left(\frac{1}{f_s}\right)^2}$ $A2 = \frac{2y\left(\frac{2}{f_s}\right) - y\left(\frac{3}{f_s}\right) - y\left(\frac{2}{f_s}\right)^2 + y\left(\frac{1}{f_s}\right)y\left(\frac{3}{f_s}\right) - y\left(\frac{1}{f_s}\right)}{Y\left(\frac{2}{f_s}\right) - 2y\left(\frac{1}{f_s}\right) + y\left(\frac{1}{f_s}\right)^2}$ $BO=O$ $B1 = y\left(\frac{1}{f_s}\right)$ $B2 = 1 - A1 - A2 - B1$
Double-Pole High Pass	$A1 = \frac{y\left(\frac{1}{f_s}\right)y\left(\frac{2}{f_s}\right) - y\left(\frac{3}{f_s}\right)}{y\left(\frac{1}{f_s}\right)^2 - y\left(\frac{2}{f_s}\right)}$ $A2 = \frac{y\left(\frac{1}{f_2}\right)y\left(\frac{3}{f_s}\right) - y\left(\frac{2}{f_s}\right)^2}{y\left(\frac{1}{f_s}\right)^2 - y\left(\frac{2}{f_s}\right)}$ $B0 = 1$ $B1 = \frac{-\left(B2 + y\left(\frac{1}{f_s}\right)^2 - y\left(\frac{2}{f_s}\right)\right)}{y\left(\frac{1}{f_s}\right)^2 - y\left(\frac{2}{f_s}\right)}$ $B2 = \frac{2y\left(\frac{1}{f_s}\right)y\left(\frac{2}{f_s}\right) - y\left(\frac{3}{f_s}\right) - y\left(\frac{1}{f_s}\right)^3}{y\left(\frac{1}{f_s}\right)^2 - y\left(\frac{2}{f_s}\right)}$
Double-Pole Band Pass	$A1\frac{y\left(\frac{2}{f_s}\right)}{y\left(\frac{1}{f_s}\right)}$ $A2 = \frac{y\left(\frac{1}{f_s}\right)y\left(\frac{3}{f_s}\right) - y\left(\frac{2}{f_s}\right)^2}{y\left(\frac{1}{f_s}\right)}$ $BO=O$ $B1 = y\left(\frac{1}{f_s}\right)$ $B2 = -B1$
Double-Pole Notch Pass (Band Stop)	$A1 - \frac{1 - y\left(\frac{2}{f_s}\right)}{1 - y\left(\frac{1}{f_s}\right)}$ $A2 = \frac{y\left(\frac{1}{f_s}\right)y\left(\frac{3}{f_s}\right) - y\left(\frac{2}{f_s}\right)^2 + 2y\left(\frac{2}{f_s}\right) - y\left(\frac{1}{f_s}\right) - y\left(\frac{3}{f_s}\right)}{\left(1 - y\left(\frac{1}{f_s}\right)\right)}$ $B0 = 1$ $B1 = y\left(\frac{1}{f_s}\right) - 1 - A1$ $B2 = -(A1 + A2 + B1)$

Figure 1 **b**—*Having the equations already solved for the* digital-filter *coefficients of* eight common *filters makes writing fhe code easier.*

Remember that the output y equals *B1x* Node 1 + *BO x* Node 0. Since *BO = 0,* then Y = *B1x* Node 1. *B1* is simply a scale factor that has nothing to do with how quickly the output reaches the final value. It simply determines the gain of the circuit. Starting with the equations in Listing 1d,I can write the code of Listing 1 e.





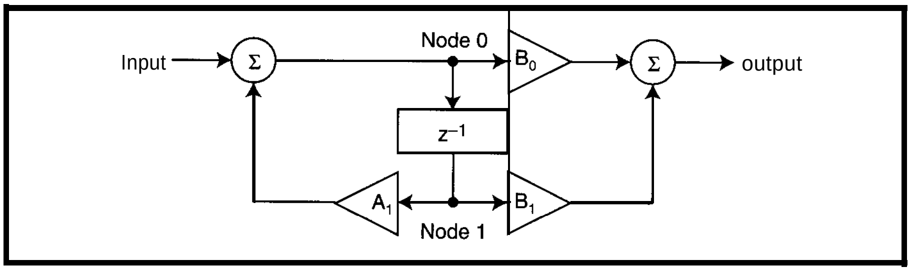


Figure 2-A common *implementation of a first-order* infinite-impulse-response *(IIR) filter uses the current sample plus the* previous *sample.*

Normally, you want GAIN to be 1, so *B1* = 1 -Al for a single-pole low-pass filter. If you experiment with different values of ***Al, you*** find that values of ***Al*** near zero reach the steady state quickly.

When ***Al*** is just under 1, it takes a long time to reach the steady state. So, ***Al*** is somehow related to the size of the RC time constant. Eventually, we need to find a simple way to compute ***Al*** from the RC time constant of a low-pass filter.

But, before we do that, let's look at a second-order filter and find a general solution that works for all first- and second-order filters.

SECOND-ORDER FILTERS

The second-order filter (see Figure 3) is a simple extension of the first-order filter. Its equations are included in Listing 2a.

Obviously, if you let A2 and B2 = 0, it reduces to the first-order case. Consider the step response of this filter. Initially, NODE-O, NODE_1, and NODE-2 are 0, so the output is 0. Table 2 shows the sequences of values appearing at the three nodes and the output.

The column on the right is of immediate interest. If we know the values of ***AI,*** A2, BO, *B1,* and B2, the column on the right lets us calculate the first four outputs of the filter and its final (steady-state] value. Listing 2b offers the equations.,

These equations are fine if we want to analyze the performance of a filter with the known constants ***AI,*** A2, ***BO,*** *B1,* and B2. However, we usually want to go the other direction. In other words, knowing the values of *y0,* ***yl,*** *y2,y3,* and ***yf, we*** have to pick the values of ***Al,*** A2, ***BO,*** *B1,* and B2 to get the desired output.

For example, suppose we need a 50-Hz second-order low-pass filter with a damping ratio (ζ) of 0.3 and a sampling frequency of 1024 Hz. For this, the step response should be:

$$y(t) = 1 - \frac{\omega_n}{\omega_d} e^{-\alpha t} \sin(\omega_d t + \phi)$$

where

$\omega_n = 2\pi \text{ x } 50$
$\zeta = 0.3$
$\omega_d = \omega_n \times \sqrt{1.0 - \zeta^2}$
$\alpha = \zeta \times \omega_n$
$\phi = \tan^{-1}(\omega_d / \alpha)$

To find the values when ***T = 0, T =*** 1⁄1024, ***T*** = 2⁄1024, ***T***= 3⁄1024, and the final value at ***T*** = ∞, use a calculator or a computer to get:

yo = y(0) = 0.00000
yl = $y(9.76562\times10^{-4})$ = 0.04396
y2 = $y(1.95312\times10^{-3})$ = 0.16208
y3 = $y(2.92969\times10^{-3})$ = 0.33181
yf = y(∞) = 1 .ooooo

This calculation produces five equations with five constants (*y0,y1, y2,y3,* and *y5*) and five unknowns ***(Al,*** A2, ***BO,*** *B1,* and B2). Since in Listing 2b, *y0* = BO, the first equation solves itself. You can substitute the constant *y0* for ***BO*** everywhere it appears in the

Listing I--The *equations used to calculate the output of a first-order infinite-impulse-response (IIR) filter can be reduced in complexity for the* final application.

```
a) NODE_1  =  NODE-0
     NODE-0 = X + Al * NODE_1
     Y = BO * NODE-0 + B1 * NODE_1

b) For  T  =  0
       X = 1 (a unit step input)
       NODE-1 = 0
       NODE_0 = X + Al * NODE-1 = 1
       Y = BO * NODE_0 + B1 * NODE-1 = BO
     therefore
     y(0) = BO

c)   For T = 1/fs (the first sample time)
       X = 1
       NODE_1 = 1 (the previous value of NODE_0)
       NODE_0 = X + Al * NODE_1 = 1 + Al
       Y = BO * NODE_0 + B1 * NODE-1 = BO*(1 + Al) + B1
     therefore
     y(1/fs) = BO*(1 + Al) + B1

d) FOR T = infinity
       NODE-0 = X + Al * NODE_1
       X = 1
       NODE-1 = NODE-0
     therefore
       NODE_0 = 1 + Al * NODE_0
     Solving for NODE-O, yields
       NODE_0 = 1 / (1  Al)
     Solving for the final output Y, we get
       Y = BO * NODE_0 + B1 * NODE-1
       Y = BO * NODE_0 + B1 * NODE-0
       Y = (BO + B1) * NODE-0
       Y = (BO + B1) / (1  Al)
     y(t = infinity) = (BO + B1) / (1  Al)

e)   GAIN = (BO + B1) / (1  Al)
     GAIN = B1 / (1  Al)
     B1 = GAIN * (1  Al)
```

remainder of Listing 2b's equations. This substitution results in four (nonlinear) equations with four unknowns. All that remains to be done is “just a little algebra.”

I spent three weekends solving these equations by hand. Each time I got a different wrong answer. Finally, I wimped out and had MathCad solve them for me (see Figure 4).

These equations aren't really as bad as they look. For all step responses of interest, *y0* and *yf* are either 0 or 1, so many of the terms drop out. In fact, for the second-order low-pass and second-order band-pass, the fourth equation simplifies to $B1 = y1$. For the second-order band-pass, the final equation simplifies to B2 = $-y1$. So, although the general solution is messy, the solutions of interest aren't too bad.

FINDING THE GOLD

How do you turn a circuit into a digital filter?

In general, just compute the values of the step response at 0, the first three sample times, and the final value of the step response. Plug these five values into the equations in Figure 4 to get the five filter coefficients. (In some cases, you need to cancel out terms or use other algebraic tricks to avoid dividing zero by zero.) I've already done the math for the eight most common filters and put the answers in Figure lb.

Let's work through an example. We already computed the step response for a 50-Hz double-pole low-pass filter with a damping ratio of 0.3. The values at the five times of interest were:

$$yo = y(0) = 0.00000$$
$$yl = y(9.76562\times10^{-4}) = 0.04396$$
$$y2 = y(1.95312\times10^{-3}) = 0.16208$$
$$y3 = y(2.92969e\times10^{-3}) = 0.33181$$
$$yf = y(\infty) = 1.00000$$

If you plug these constants into the digital-filter-coefficient equations for the double-pole low-pass filter shown in Figure 1 b, you end up with a value of 1.746493 for ***AZ***, -0.831797 for A2, 0.0 for *B0*, 0.04396 for *B1*, and -0.041344 for B2.

Sample	Input	Output
-2	0	0.00000
-1	0	0.00000
0	1	0.00000
1	1	0.50000
2	1	0.75000
3	1	0.87500
4	1	0.93750
5	1	0.96875
6	1	0.98438
7	1	0.99219
8	1	0.99609
9	1	0.99805
10	1	0.99902

Table 1—*In this step response of a single-pole digital filter, A1 = 0.5, B0 = 0, and B1 = 0.*

PRACTICAL LIMITATIONS

This all sounds great in theory, but you might be surprised with the results if you pushed the filter to its limits. Of course, this implies you know what its limits are-which most people don't.

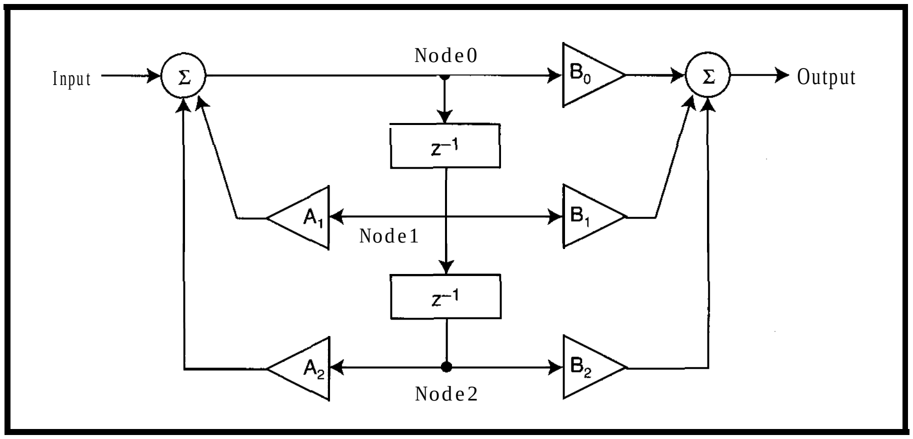


Figure 3-A *common implementation of a second-order infinite impulse response (IIR) filter uses the current sample in addition to the previous two in ifs calculations.*

Let's begin with the sampling rate because it is the one most likely to bite you. Many people believe that Nyquist said it is adequate to sample signals of interest at least twice per cycle. However, Nyquist didn't say that. Nyquist said you are completely out of luck if you sample less than twice per cycle. If you sample at exactly twice per cycle, Nyquist said you are only in quicksand up to your ears. It is theoretically possible to extract the data, but it requires an ideal brick-wall filter-something which exists only in theory.

Sample	X	N0	N1	N2	Y
-2	0	0	0	0	0
-1	0	0	0	0	0
0	1	1	0	0	BO
1	1	$1+A1$	1	0	$B0\times N0+B1$
2	1	$1+A1+A1^2+A2$	$1+A1$	1	$B0\times N0+B1\times N1+B2$
3	1	$1+A1+A1^2+A1^3+2\times A1\times A2+A2$	$1+A1+A1^2+A2$	$1+A1$	$B0\times N0+B1\times N1+B2\times N2$
	1	1-Al-A2	1-Al-A2	1-Al-A2	$(B0+B1+B2)/(1-A1-A2)$

Table 2—*These* equations calculate *the output and* infernal *nodes when a step input is applied to a second-order filter.*

In practice, sampling at less than eight times per cycle adds so much harmonic distortion to the reconstructed output signal that you have to provide serious analog low-pass filtering on the output. Filtering in the digital domain doesn't help because the distortion is introduced by the sample-and-hold nature of the output of the D/A converter, which occurs after the digital filtering. The digital filter won't remove the distortion because it isn't introduced until after the digital filter processes the data.

If noise on the analog input exceeds half the sampling frequency, that noise is heterodyned down to an alias frequency less than half the sampling frequency by the input conversion process. Since it appears to be a much lower frequency than it really is, you can't remove it with a digital low-pass filter. Instead, you must have an analog antialiasing filter before the A/D converter that reduces all noise above half the sampling frequency to an amplitude less than the resolution of the A/D converter. If you don't, that high-frequency noise is indistinguishable from low-frequency signals.

In practice, this means the higher sampling frequency is better. With higher sampling frequencies, you can use a simpler analog-antialiasing filter on the input and a simpler analog-reconstruction filter on the output.

It is a good idea to compute the cut-off frequency as a fraction of the sampling frequency. Any time the cut-off frequency is more than half the sampling frequency, the filter is theoretically absurd. Since there are no frequencies higher than half the sampling frequency present, there is nothing to filter out!

Any time the cut-off frequency is more than one quarter of the sampling frequency, a single-pole filter is practically useless because the out-of-band data is only reduced by 3 dB at the most. So, why bother? As an absolute minimum for a single-pole filter, the sampling frequency ought to be at least eight times the cut-off frequency to do any good at all.

INTERNAL MAGNIFICATION

The values of the variables NODE_0, NODE_1, and NODE-2 can reach values as high as 1 / (1 -Al – A2) times the input. This usually isn't a problem if you use floating-point arithmetic, but it may be a problem if you use integer arithmetic.

Suppose you design a single-pole low-pass filter with a cut-off frequency that is 1/50 of the sampling frequency. Al is 0.98, so the internal magnification is 50.5. Suppose you use a 16-bit processor and don't want to use double-precision arithmetic. With a 12-bit A/D converter, the input is in the range ±2048. This range is internally magnified to ±103424, which exceeds the range of 16-bit arithmetic. To keep the internal nodes from overflowing, you have to limit the input to ±648. So, save your money and use a 10-bit A/D converter, which limits the input to ±5 11.

You still might need to worry about computational errors when you multiply the internal nodes by the *B* coefficients to compute the output. However, since *B* coefficients tend to be less than 1, underflow is more often a problem than overflow.

If the single-pole low-pass filter we have been using for an example has unity gain, then *B1* = 0.0198. You can factor that into 0.015625 + 0.0039062 + 0.0002687 and ignore the last term. You can multiply this value by 0.015625 by shifting right six times and multiply by 0.0039062 by shifting right eight times. The output value is then reduced to the range ±5 11, so there is no point in using a D/A converter with more than 10 bits.

Of course, you could pick *B1* to be 64 x 0.0198 and adjust the output of the D/A converter to $\frac{1}{64}$ of normal to get more apparent resolution. However, if the input was quantized to 10 bits, then the accuracy of the output can't be any better than 10 bits.

A BALANCING ACT

On one hand, you want the sampling frequency to be as high as possible (to reduce distortion, phase errors, and avoid aliasing). But, the sum of ***AI*** + A2 approaches 1 as sampling frequency goes up. The internal magnification is 1/(1 – (*Al* + *A2*}), so the internal magnification problem skyrockets at higher sampling frequencies.

As the sampling frequency goes up, internal magnification may force you to use 32- or 64-bit arithmetic, and you will have less time between samples to do the arithmetic. Use good judgment to pick an appropriate sampling frequency.

TEST THE FILTERS

When you build an analog filter you sometimes discover that the circuit you build does not perform exactly as the theory predicts. The stray capacitance and internal resistance you neglected in your calculations isn't neglected by the electrons flowing through the wires. The cut-off frequency is almost correct, and the phase shift is within a few degrees of what you expect, but you usually have to tweak the filter a little bit to get it just right.

You might be surprised to discover that digital filters can be just as touchy as analog filters. In fact, some digital filters can be *very* sensitive to small changes in coefficient values. Digital filters are susceptible to the "small differences between big numbers" problem that has long been recognized in numerical analysis.

For example, in the double-pole low-pass example discussed earlier, *B1* = 0.04396 and *B2* = -0.041344. Since NODE_1 often has nearly the same value as NODE-0, the output is the difference between two nearly identical values. Inadequate precision or accuracy results in noticeable errors.

Looking back over the calculations used to find *Al* and A2 for that filter, you can see that the calculations derived from the quotient of a series of additions and subtractions. Common sense tells you that if you use too few digits, you won't get very accurate values for *AZ* and A2.

Listing 2-These *equations compute the* output *of a second-order digital filter.*

```
a)  NODE_2 = NODE_1
    NODE_1 = NODE_0
    NODE_0 = X + Al * NODE_1 + A2 * NODE_2
    Y = BO * NODE_0 + B1 * NODE_1 + B2 * NODE-2

b)  y0 = BO
    yl = B0+B0*A1+B1
    y2 = B0+B0*A1+B0*A1^2+B0*A2+B1+B1*A1+B2
    y3 = B0+B0*A1+B0*A1^2+B0*A1^3+B0*2*A1*A2+B0*A2+B1+B1*A1+
           B1*A1^2+B1*A2+B2+B2*A1
    yf = (B0+B1+B2)/(1-A1-A2)
```

If you're using integer arithmetic, you might be tempted to change the value of a coefficient slightly to simplify the arithmetic. For example, you might want to change 0.9377 to 0.9375 because 0.9375 = $1 - 1/16$. Even though you're only changing the coefficient by 0.02%, it significantly changes the cut-off frequency or out-of-band gain.

As you can see, there is a lot that can go wrong, even if you pick the coefficients correctly. For this reason, it is important to test the digital filter carefully to ensure it is behaving properly. Hit it with a low-frequency square wave and measure the step response. Apply sine waves at several frequencies and verify that the gains and phases are correct. Apply white noise (up to half the sampling frequency) and look at the output on a spectrum analyzer to verify that the filter gives the proper response.

If the test results don't surprise you a little, you didn't test the filter well enough. The filter will not perform exactly like its analog coun-

$$A1 = -\frac{-y0yf + y1yf - y3yf + y2yf + y0y3 - y1y2}{y0yf + y2yf - 2y1yf + y1y1 - y0y2}$$

$$A2 = -\frac{y1yf - 2y2yf + y3yf + y2y2 - y1y3}{y0yf + y2yf - 2y1yf + y1y1 - y0y2}$$

$$B0 = y0$$

$$B1 = \frac{4y0y1yf + y1y2yf - 2(y1)^2yf + (y1)^3 - 2y0y1y2 - 2(y0)^2yf - y0(y1)^2 + (y0)^2y2 - y0y3yf + (y0)^2y3}{y0yf + y2yf - 2ylyf + (y1)^2 - yoy2}$$

$$B2 = \frac{2y0y1y2 + 2y0y3yf - y1y3yf + (yo)^2yf - (yo)^2y3 + (y2)^2yf - 2y0y2yf + 3(y1)^2yf - 2y0y1yf - (y1)^3 - 2y1y2yf}{y0yf + y2yf - 2ylyf + (y1)^2 - y0y2}$$

Figure 4-Given *the first four output* values *(y0–y3) and the final value (yf) of the desired step response of a* **digital filter,** *you can compute the digital filter coefficients A1, A2, B0, B1, and B2 using* ***these equations.***

terpart, especially in the frequency range of $F_s/4$ to $F_s/2$. The sample-and-hold function introduces harmonic distortion that transfers some of the power from the fundamental frequency to higher frequencies. The actual gain may be slightly lower than calculated at some frequencies.

The sample-and-hold also introduces a delay that shows up as a few extra degrees of phase lag for some frequencies. When you test with a single-frequency sine wave above $F_s/4$, you see some amplitude modulation as the sampling frequency "beats" against the test signal frequency. (Amplitude is a function of sampling phase at lower sampling rates. As the samples occur at different phases, the amplitude changes.)

I've given you the universal formula to digital filter alchemy, but I must warn you to check the quality of your gold. You've got all my lab notes, but it's up to you to become great alchemical wizards. ❑

Do-While Jones 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.*

IRS

401 Very Useful
402 Moderately Useful
403 Not Useful

FEATURE ARTICLE

Sébastien Roy

Speech Compression Techniques and the CDV-1 Digital Voice Box

> None of the methods for storing digitally recorded speeceh met project needs. After summarizing various methods, Sébastien refines the companded differential PCM method, a solution which fit the bill.

I started with the idea of designing a digital-voice-box-circuit-you know, one capable of storing and playing half a minute of digital sound. The list of applications (e.g., talking posters, door bells, etc.) is endless for a device that stores sounds in a small physical space. For instance, imagine greeting a trespasser on your property with a loud, "No trespassing. Be gone Intruder! "

I wanted the device to be commercially viable. The features and cost needed to be competitive with similar products. Most importantly, the device had to be as small as possible.

With these considerations in mind, I came up with a set of specifications. It needed:

- a sound coding method to achieve 2: **1** compression over 8-bit PCM (with this, a 64-KB EPROM to hold 16 s of sound sampled at 8 kHz},
- to generate only frequencies below 1.7 MHz, eliminating the need for FCC or DOC approval (DOC is the Canadian equivalent to the FCC),
- to be compact for use in space-conscious applications, and
- sound creation for the voice box using a personal computer without additional, expensive hardware.

Digital sound data contains a lot of redundancy and is compressed in a number of ways. My challenge was to find a compression method powerful enough to produce a 2:1 gain and simple enough for a microcontroller to run on **1.7** MHz.

I decided to let the microcontroller handle the signal processing tasks without specialized hardware. I wasn't sure this was possible, but dedicated speech processing chips significantly add to project cost and impose a specific coding scheme such as ADPCM or CVSD. Also, I wanted to find out how much a low-end microcontroller could accomplish.

At that point, I was pretty sure I had to sacrifice sound quality to meet these constraints. Nonetheless, I set out to design the perfect voice box: compact, cheap, elegant, simple, and "FCC proof." I was in for surprises on the long road to a prototype.

THEORY OF SPEECH COMPRESSION

First, I reviewed the theory of speech compression methods and algorithms to find something that would fit the bill. My research and reflection process produced interesting insights. I eventually created a new speech-compression algorithm. I'll start by outlining the essential concepts of speech-coding theory.

The advantages of representing sound or speech digitally are well known. Digital sound representation makes possible encryption for privacy. Digital data is more resistant to degradation than analog recordings and is easier to manipulate. Finally, error-correction systems work only with digital sounds for transmission over a noisy channel.

Conventional analog-to-digital conversion techniques create large digital files. For example, in digital telephony the bandwidth ranges from 0.3 to 3.3 kHz. The sampling rate is usually 8 kHz with a resolution of 12 bits. Accordingly, a single second of telephone-quality speech can occupy as much as 96,000 bits or 11.7 KB.

However, speech signal redundancy makes it possible to encode speech more efficiently. The compression method must be carefully selected to provide an adequate balance between sound quality, compression ratio, and computational complexity. If we choose the signal-to-noise (S/N) ratio as a measure of sound quality, this equation is handy:

$$S_{dB} = 6B + 4.8 - 20\log_{10}h \qquad (1)$$

where S_{dB} is the S/N ratio in decibels, B is the number of bits per sample, and h is the headroom factor (i.e., a safety margin to prevent saturation or clipping, usually set to 4). This formula confirms that 12 bits are required for a S/N ratio of 60 dB with a headroom factor of 4.

Speech signals present a coherent structure, which lends to compression efforts. Two characteristics are useful for simple compression algorithms.

First, speech is concentrated in the low amplitudes, which means that statistically the smaller sample values occur more often than the larger ones. Compression methods like companding and Huffman encoding take advantage of this characteristic.

Second, there is a high correlation between neighboring samples because speech parameters (pitch, amplitude, etc.) vary slowly in time. ADPCM and CVSD take advantage of this.

It is important to distinguish between speech encoders and speech transcoders. Speech encoders convert analog speech into a given digital representation. Coding schemes like ADPCM and CVSD work directly from the analog signal.

Speech transcoders convert digitally encoded speech to another form of digital speech. For example, Huffman encoding requires a PCM digital sound file as input.

Now, let's move on to look at various types of compression.

COMPRESSION BASED ON SPEECH DENSITY

Speech signals are heavily concentrated in the low amplitudes, which means the probability density of speech is not uniform. In fact, speech approximates a modified gamma density function. Hence, a uniform A/D quantizer does not yield the optimal S/N ratio. It is much better to use a nonuniform quantizer in which the steps are smallest at lower levels. The step sizes in the optimal quantizer are adjusted to make each sample value equally likely.

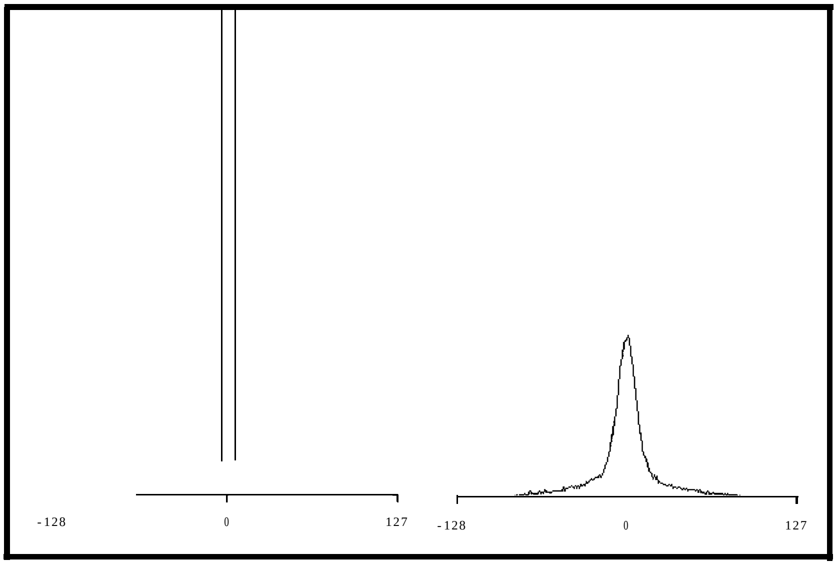


Figure 1—*The probability density functions of DPCM (left) and PCM (right) are derived from a small sound data set Observe that the* concentration in *the low* amplitudes is *much more pronounced in DPCM than in PCM*

A nonuniform quantizer usually consists of a nonlinear distortion filter followed by a uniform quantizer. The filter compresses the dynamic range of the signal to make its probability density as close to uniform as possible. p-law and A-law are the two distortion curves often used for these filters.

The process is called *companding* (for compressing/expanding) and is heavily used in digital telephony. Standard 8-bit u-law cornpander chips produce roughly the same sound quality as a 12-bit uniform quantizer.

Companding gives a compression ratio of roughly 3:2 and its principles are fairly straightforward. But, how can it be applied to our voice box? The obvious way is to incorporate a dedicated 8-bit u-law codec chip to act as a nonlinear D/A converter.

This solution, however, has two drawbacks. Codec chips tend to be expensive. In addition, they are designed to give 12-bit quality in 8 bits. What we really want is 8-bit quality in 4 bits! (Some codecs compress 8 bits into 6 bits, but they're still expensive.)

Of course, it is possible to implement companding in software with little overhead. However, while companding holds promise, it does not reach the target 2:1 compression ratio by itself.

Huffman encoding uses variable-size words to represent samples. Short 2-, 3-, or 4-bit words represent the most frequent values while longer (up to 12-bit) words represent the rarest values. Unfortunately, this method is too complex for our purposes because the words are of unequal size.

COMPRESSION BASED ON AUTOCORRELATION

Most popular voice-compression schemes take advantage of the high correlation between successive samples in speech signals. For example, differential PCM (DPCM) encodes the difference between samples instead of the samples themselves.

While this method does not offer significant compression, its data has a much lower average power than standard PCM. In other words, a DPCM data set should be easier to compress than the equivalent PCM data set due to its reduced variance.

Incorporating a prediction filter in the encoding process improves the basic DPCM concept. Such a filter embodies knowledge of the statistical properties of speech signals in its coefficients and predicts the value of an unknown sample based on previous samples.

The difference between a sample and its predicted estimate is encoded.

Using a second-order predictor, the variance of the system is reduced by 6 dB. According to Equation 1, a reduction of 6 dB implies that we can use one less bit than PCM in quantizing the residual and still maintain the same S/N ratio.

We can improve the performance of the prediction filter by making it adaptive. The resulting speech compression algorithm, adaptive DPCM (ADPCM), is widely used in telephony and computer-based speech.

If a little sound degradation is acceptable, ADPCM provides a compression ratio of 2: I, but it is complex. ADPCM is typically implemented using dedicated ICs or DSPs.

As far as our voice box is concerned, any type of prediction scheme is difficult to implement in real time because of the numerous multiplication operations involved.

Interval	CDPCM Symbol	Decoded DPCM Value
-128 to -19	0	-40
-18 to-10	1	-13
-9 to -6	2	-7
-5, -4	3	-4
-3	4	-3
-2	5	-2
-1	6	-1
0	7	0
1	8	1
2	9	2
3	10	3
4, 5	11	4
6-8	12	7
9-14	13	13
15-24	14	19
25-127	15	40

Table 1-A *standard CDPCM look-up table is used for decoding.*

Simpler differential modulation schemes include delta modulation (DM), which is really DPCM with a 1-bit quantizer. While this is an easy modulation scheme to implement, it is difficult to obtain acceptable speech quality with DM. Even at a sampling rate of eight times the Nyquist frequency, the S/N ratio is only 20 dB. In other words, even at bit rates comparable to PCM, the sound quality leaves much to be desired.

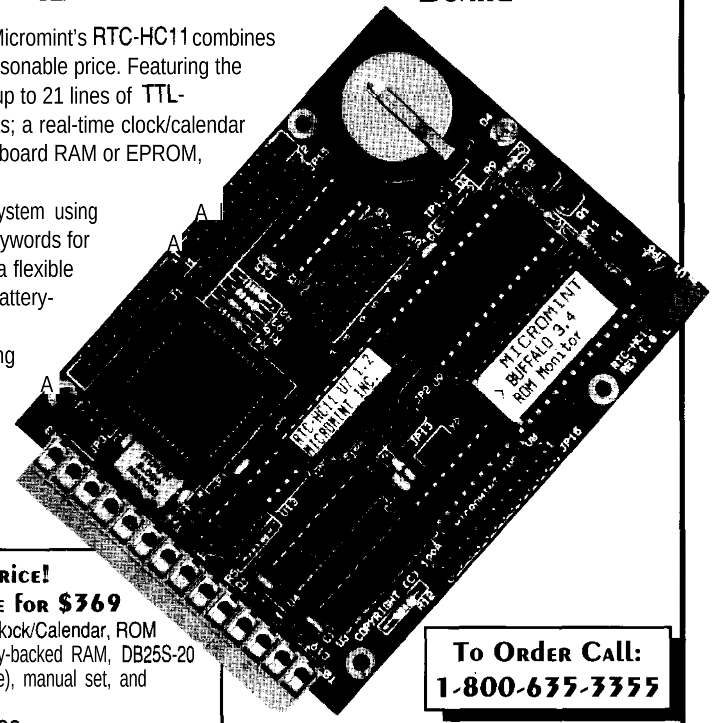
Adaptive delta modulation schemes fare much better by adapting the quantizer's step size as a function of recent sample values. Continuously Variable Slope Delta (CVSD) modulation is an adaptive system which uses a I-bit quantizer.

A sample value of 1 means the output should be increased by the current step size while a 0 means the output should be decreased accordingly. CVSD attempts to vary the step size to minimize the occurrence of slope overload and granular noise.

Slope overload occurs when the slope of the analog signal is so steep that the encoder can't keep up. This effect can be minimized or prevented by enlarging the step size sufficiently.

Granular noise occurs when the analog signal is constant. The CVSD system has no symbols to represent steady state, so a constant input is represented by alternating ones and zeros. Accordingly, the effect of granular noise is minimized when the step size is sufficiently small.

The adaptation strategy used by CVSD is fairly simple. The previous three samples are examined. If they are



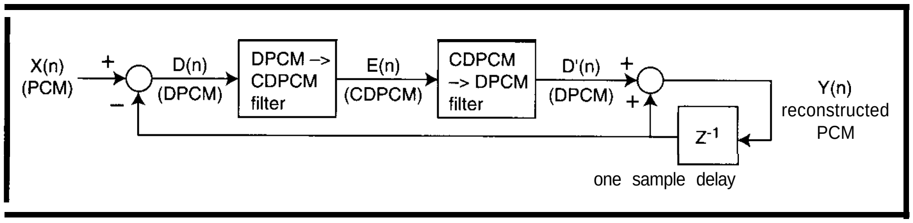


Figure 2—*The* CDPCM encoder uses the reconstructed *PCM* signal Y(n) to avoid cumulative errors in *the CDPCM datastream* E(n).

identical, it indicates slope overload. The step size is therefore increased by a multiplicative constant and an additive constant.

If the three samples are not identical, the step size is progressively reduced. It is important to carefully choose the values of the constants used to adjust the step size so the effects of both slope overload and granular noise are minimized.

To achieve adequate speech quality with CVSD, the sampling frequency should be four times higher than with PCM. Consequently, CVSD systems often work at 32 kbps, which is half the bit rate required by an 8-bit, 8-kHz PCM system.

CVSD proves that a relatively simple algorithm can achieve significant compression gains over standard PCM. Nonetheless, there are a number of reasons why CVSD does not fit the design constraints outlined earlier.

CVSD requires a lot of bit-level manipulation. It also involves some multiplication in the step-size adjustment procedure. Despite its conceptual simplicity, CVSD might still be too complex to be implemented on a simple controller running at 1.7 MHz.

Furthermore, the sound quality is better if the analog signal is directly coded into CVSD. The constraints outlined in the beginning specify the use of a PC to create the sounds, so the analog signal is not available for direct coding.

Of course, a simple software transcoder can be written to convert between the PCM data typically produced by computer sound hardware and CVSD. However, the transcoding process results in a loss of quality because the quantization noise introduced by PCM is compounded by the noise inherent to CVSD.

Indeed, PCM introduces quantization noise because it is limited to a discrete number levels-256 in the case of an 8-bit system. CVSD has no such limitation. In fact, dedicated CVSD chips typically use 10 bits internally to represent voltage levels.

However, CVSD introduces slope overload and granular noise because of its adaptive nature. When using a transcoder, both PCM and CVSD noise are present and their combined effects further deteriorate sound quality.

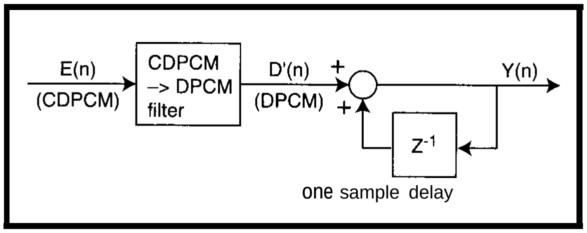
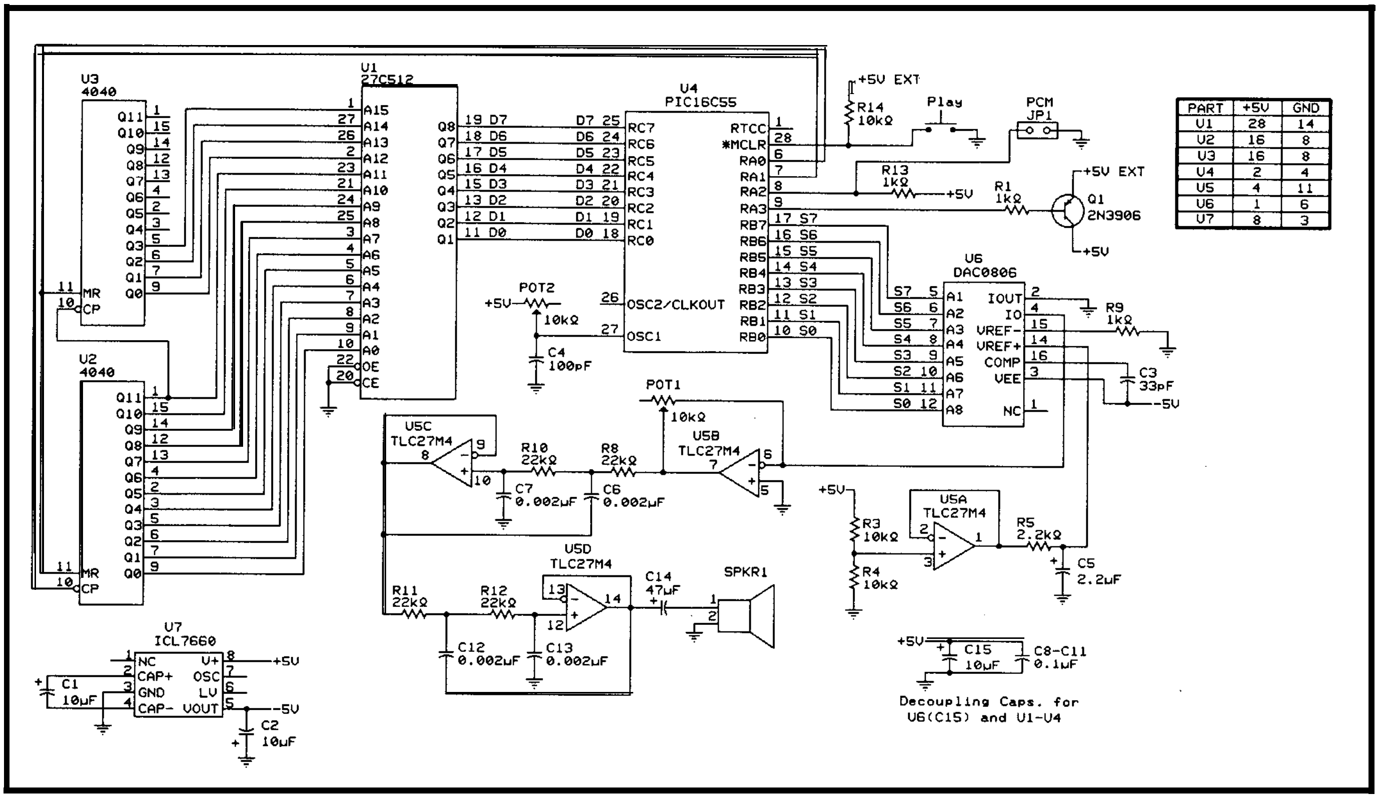


Figure 3—*The CDPCM* decoder *is simple-only a look-up table and an* addition are *necessary to decode a sample.*



PART	+5V	GND
U1	28	14
U2	16	8
U3	16	8
U4	2	4
U5	4	11
U6	1	6
U7	8	3

Figure *4-The CDV-1 is composed of* a *PIC16C55 microcontroller, an EPROM, counters, a D/A converter, and a simple analog section that provides amplification and filtering.*

Having examined all these speech digitizing methods, it became painfully obvious to me that none of them, not even CVSD, met the constraints of the voice-box project.

However, I remained convinced that sound signals contain enough inherent redundancy to provide a solution. It seemed possible to borrow concepts from existing speech compression-methods to create a new method specifically for this project.

Listing 1—*This pseudocode works through the algorithm for the playback of PCM with oversampling.*

```
1  read a sample from the EPROM
2  is it equal to 255? If so, go to sleep
3  copy sample to 255? If so, go to sleep
4  increment counters to update EPROM address
5  wait 14 cycles
6  take current value on D/A converter port and add it to
     next sample value from EPROM
7  divide by two using the "rotate right" instruction
8  copy this estimated sample to D/A converter
9  wait 14 cycles
10 go back to step 1
```

COMPANDED DIFFERENTIAL PCM

I needed an algorithm that exploited redundancy in sound signals and required no more than a couple of additions and/or table lookups to decode a sample. Only companding seemed simple enough to implement in software. Unfortunately, its compression ratio is less than satisfactory.

The probability density curves in Figure 1 reveal some statistical aspects of a differential PCM data set compared to ordinary PCM. DPCM exhibits more redundancy. Of the 256 possible values, 16 account for over 65% of the samples. It should therefore be possible to compress a DPCM data set more efficiently than PCM using Huffman encoding or companding.

But how do we apply a concept like companding to differential data? As you may recall, p-law and A-law companding distort the sound signal giving it a uniform probability density. The goal is to make each sample equally likely.

I propose a new speech compression method called ***CDPCM*** (Companded Differential PCM) which maps the 256 possible DPCM sample values to 16 equally likely symbols. The computer generates a DPCM probability curve based on a large, representative speech data set to construct a standard look-up table.

The curve is divided into 16 intervals of equal likelihood. Each interval is mapped to one of 16 symbols, which means only 4 bits are required to encode a sample. We thus achieve 2: 1 compression.

A symbol is decoded by replacing it with the most likely value in the interval it represents (see Table 1). This introduces error in the reconstructed speech signal, but the overall impact on speech quality is minimal.

While there are only 16 symbols, the reconstructed PCM samples from a CDPCM decoder can take any value from 0 to 255. The difference between adjacent samples is limited to 16 values.

A detailed statistical analysis of the noise introduced by this algorithm is beyond the scope of this article. Suffice it to say that CDPCM-induced noise concentration is in the high frequencies and can be virtually eliminated by playing it through an appropriate filter.

When a decoded CDPCM signal plays back without a filter, noise appears only for plosive sounds (e.g., if you say "p," the burst of air on your hand marks it as a plosive sound). When it plays back through an appropriate low-pass filter, the difference in quality with PCM is nearly inaudible.

The process of reconverting data to PCM at the decoding end is like an integrator. Accordingly, any error we introduce in the DPCM data is integrated to infinity. The encoder tracks the error introduced and compensates, avoiding this problem. This is accomplished by using the reconstructed PCM signal Y(n) (the signal obtained by decoding the CDPCM data set) in the encoding process (see Figure 2).

Ordinarily, the DPCM signal ***D(n)*** is derived by using:

$$d[n] = x[n] - x[n\text{-}I] \qquad (2)$$

The CDPCM signal ***E(n)*** is obtained simply by passing ***D(n)*** through the look-up table. At the decoding end, the reconstructed DPCM signal ***D'(n)*** is likewise obtained by passing ***E(n)*** through the opposite look-up table. This equation produces the decoded PCM output:

$$y[n] = y[n\text{-}I] + d'[n]$$

To compensate for error, Y(n) is used as a reference for deriving ***D(n).*** Equation 2 must then be replaced by:

$$d[n] = x[n] - y[n-1]$$

This equation implies that the encoder includes a simulated decoder. In fact, as Figure 3 indicates, the encoder is more complex than the decoder. Only the decoder needs to be embedded in the voice box.

The encoder must still address the possibility of overflow. The error introduced by the encoder may bring *Y(n)* above 256 or below 0. The possibility of overflow must be verified by the encoder and dealt with by replacing the offending symbol by a smaller one.

DIGITAL VOICE BOX

The CDV-1 circuit is made up of a microcontroller with the CDPCM decoding algorithm embedded in its firmware, an EPROM containing the actual sound data, a standard 8-bit D/A converter, and a simple analog filter and amplifier circuit.

You only need an external S-V power source, a push-button switch or TTL activation signal; and a speaker. Activating the switch triggers the playback of the sound data. At the end, the circuit goes into low-power sleep mode until the next request.

Listing 2—*A simple C program generates a raw CDPCM file from PCM data contained in an AIFF file.*

```
#include <stdio.h>

unsigned char encode(int *d);

main()
{
  FILE *old, *new;
  int c, i, d, mo;
  unsigned char nibble[2];
  char table[16] = {-40,-13,-7,-4,-3,-2,-1,0,1,2,3,4,7,11,19,40};
  old = fopen("filename of source (PCM) sound file", "rb");
  new = fopen("CDPCM Output filename", "wb");

/* The following section should be used only with AIFF files */
  i = 0;
  while (i == 0) {           /* finding the 'SSND' chunk */
    c = fgetc(old);
    if (c == 'S') {
      c = fgetc(old);
      if (c == 'S') {
        c = fgetc(old);
        if (c == 'N') {
          c = fgetc(old);
          if (c == 'D') {
            c = fgetc(old);
            i = 1;
          }
        }
      }
    }
  }
  for (i = 0; i < 12; i++)  { /* skip the header */
    c = fgetc(old);
  }
/* End of AIFF section */
  mo = 128;           /* initial value of the running sum is 0=20
                      in offset binary, that becomes 128 */
  i = cnt = 0;
  while ((c = fgetc(old)) != EOF) {
    if (c < 128)             /* Convert 2's complement to */
      c += 128;              /* binary offset */
    else
      c -= 128;

    d = c - mo;              /* Generate DPCM sample */
    nibble[i] = encode(&d);
    while (mo + d > 255)     /* correct overload conditions */
      d = table[--nibble[i]];
    while (mo + d < 0)
      d = table[++nibble[i]];
    if (i == 0)      /* Combine two nybbles, output a byte */
      i = 1;
    else {
      i = 0;
      fputc((unsigned char)((nibble[1] << 4) | nibble[0]), new);
    }
    mo += d;                 /* update running sum */
  }
  fputc(EOF, new);           /* close files */
  fclose(old);
  fclose(new);
}

/* Convert from DPCM to CDPCM by converting values within spe-
cific intervals to one of sixteen equally likely symbols. The
value of d is modified to serve as reference in the main loop
and thus compensate the cumulative effect of errors */
```

(continued)

Listing **2**—*continued*

```
unsigned char encode(int *d)
{
  unsigned char res;

  switch (*d){
    case -5:
    case -4: *d = -4; res = 3;
             break;
    case -3: res = 4;
             break:
    case -2: res = 5;
             break;
    case -1: res = 6;
             break;
    case 0:   res = 7;
             break:
    case 1:   res = 8:
             break;
    case 2:   res = 9;
             break:
    case 3:   res = 10;
             break:
    case 4:
    case 5:   *d = 4;  res = 11
             break;
    default: if (*d <= -19){res = 0; *d = -40;}
             else
               if (*d <= -10 && *d > -19){res = 1; *d = -13;}
               else
                 if (*d <= -6 && *d > -10) {res = 2; *d = -7;}
                 else
                   if (*d <= 8 && *d > 5){res = 12; *d = 7;}
                   else
                     if (*d <= 14 && *d > 8){res = 13: *d = 11;}
                     else
                       if (*d <= 24 && *d > 14){res = 14; *d = 19;}
                       else
                         if (*d >= 25){res = 15; *d = 40;}
             break:

  return(res);
```

The microcontroller choice was not difficult. The 87C751 was a serious candidate, but the only low-cost micro capable of handling the required load with a clock speed of 1.7 MHz was the PIC16C55. This suitability is largely because of the RISC design of the PIC family.

Most instructions execute in only one clock cycle. Because the timing of the instructions is so uniform, it also facilitates development of time-critical firmware. Absolutely indispensible for this project is the PIC's unique SWAP instruction, which interchanges the high and low nybbles within a byte in one clock cycle.

CDV-1 FEATURES

If the microcontroller is running at 1.376 MHz, the CDV-1 is capable of playing back digitized speech at a sampling rate of 8 kHz and a resolution of 8 bits while oversampling. Indeed, the firmware performs interpolation to bring the effective oversampling rate to 16 kHz. Besides improving sound quality, this interpolation also eases filter requirements.

The circuit uses a 27C5 12 EPROM which holds either CDPCM data or standard PCM data. A jumper on the board provides the desired sound format. The total current consumption is less than 50 mA during playback and less than 4 µA in sleep mode. The circuit works with any 4-6-V supply.

An RC oscillator clocks the PIC microcontroller to keep the cost down. With this type of oscillator, the operating frequency varies significantly from one PIC to another.

An adjustable RC oscillator with a 10-kΩ potentiometer is incorporated in the design because the sound's playback speed is a function of the oscillator frequency. The user can thus adjust the operating frequency from 0.7 to 2.5 MHz until the playback speed is adequate. This feature enables the playback of sounds sampled at more than 8 kHz (e.g., up to around 14 kHz).

Adjust the oscillator for adequate playback of 8-kHz sounds (around 1.4 MHz) and then fix it in place by taping the potentiometer or by gluing it with epoxy. The CDV-1 also includes a volume control potentiometer in the analog section which should be adjusted carefully during playback to get maximum output power without saturation. You should be able to get up to 0.5 W with an 8-Ω speaker and up to 1 W with a 4-R speaker.

HARDWARE OVERVIEW

The CDV-1 (see Figure 4) is built on a 2$\frac{5}{8}$″ x 2$\frac{15}{16}$″ circuit board. (The board has recently been redone on a 2 x 3½″ format.) It is as compact as possible for assembly. The ICs are in DIP packages and are relatively inexpensive and easy to find.

The microcontroller reads encoded sound data from the EPROM through port C and sends its output to the DAC0806 D/A converter through port B. Two 4040 12-bit ripple counters are cascaded to generate the addresses for the EPROM. The EPROM scan rate is controlled by the microcontroller through line RAO which is tied to the counters' clocks.

RA3 controls transistor Q1, which turns power on and off to the rest of the circuit. When the CDV-1 is not active (e.g., it is not playing a sound), the microcontroller is in low-power sleep mode, and the other components of the circuit are completely powered off for minimum current consumption. A low level on the PIC's reset line wakes it up, turns on Q1, and triggers the sound-playback sequence.

The state of line RA2 tells the microcontroller whether to work in

PCM or CDPCM mode. R15 is tied to +5 V making CDPCM the default operating mode. To operate in PCM mode, a wire jumper must be added to the circuit board between RA2 and a nearby ground trace.

The DAC0806 is a garden-variety D/A converter, widely available and inexpensive. It requires a negative reference voltage provided by the inverter made up of U7, Cl, and C2.

The D/A converter's analog output is piped to a three-stage low-pass filter/amplifier circuit built around a TLC27M4 quad op-amp. Its efficient, low-power, rail-to-rail operation and the fact that it operates from a 5-V supply make the TLC27M4 an ideal choice for this application.

The first stage consists of U5b, R6, and potentiometer R7. It serves as an adjustable amplifier controlling the sound output's volume. The first stage also converts the D/A converter's current output to voltage.

The second and third stages are two second-order low-pass filters with cut-off frequencies of 3.62 kHz. Together, they form an efficient fourth-order low-pass filter. Because the software performs oversampling, this is more than adequate to handle quantization noise. Finally, U5a provides a stable, filtered, 2.5-V reference to the D/A converter.

THE FIRMWARE

The firmware includes 132 carefully chosen instructions and handles the playback of PCM or CDPCM data in real time. The initialization code turns the circuit on, resets the counters, and verifies the state of line RA2 to determine whether it should jump to the PCM or the CDPCM code section.

The PCM section reads bytes from the EPROM and directly outputs them to the D/A converter. Between each sample pair, an estimated sample is computed. Listing 1 shows the broken-down algorithm in pseudocode.

The PCM data must be in "offset" format (not two's complement) because the software copies the samples directly from the EPROM to the D/A converter. Furthermore, the data must not contain the value 255 because it indicates the end of the sound data and shuts off the circuit.

If the sound has been properly recorded (i.e., the data has not been "clipped" because of excessive volume), it is highly unlikely that it contains a 255. Otherwise, a software filter can be written to replace all occurrences of 255 by 254.

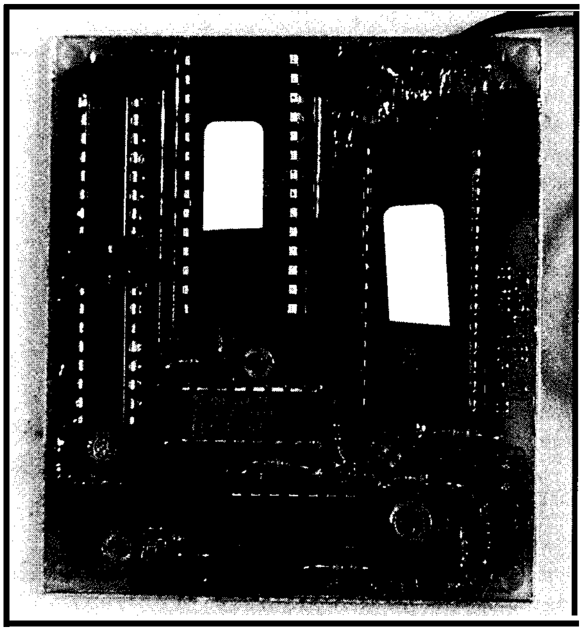


Photo 1—The CDV-1 board is about as small as if can qet usinq DIP-style components.

The CDPCM code section is more involved. The code must be very tight to provide real-time playback of 8-kHz sounds at a clock speed of only 1.376 MHz. This is possible because CDPCM offloads the bulk of the work to the encoder.

If the micro detects an overflow condition in CDPCM mode, it assumes the sound data has ended and goes to sleep. Properly encoded CDPCM data never causes an overflow, but a string of 255s in the blank portion of the EPROM will.

PREPARING SOUND DATA

How do we create sounds and incorporate them in the CDV-1 voice box? The simplest way is through a computer capable of recording sounds (e.g., most Mac models, PCs with sound cards, etc.).

You can also get prerecorded sound files off a BBS or the Internet. It should then be easy to convert these samples to PCM format. If your software has an uncompressed sound format, it is probably PCM. Sun workstations are a special case because their sound hardware includes a u-law cornpander.

Sound software use WAV, AIFF, AU, SND and other file formats to store sounds. You need to find out enough about the sound format you are using to remove headers and other irrelevant information, leaving only the PCM data.

Then you need to know whether the PCM data is in two's complement format or offset binary. There are some interesting freeware and shareware offerings out there that enable you to modify, edit, and convert sounds between formats.

Listing 2 shows how a simple CDPCM encoder can be written in C. It is written on a Macintosh running Think C, but it should compile without modifications on most platforms.

The software is designed to work with AIFF files, a sound file format commonly used on Macs. The AIFF format is quite complex and involves a number of "chunks." The initialization portion scans the file from the beginning until it finds the header indicating the start of the sound data chunk. It then skips the header and encoding begins.

Each byte converts from two's complement to offset binary before being encoded as a 4-bit nybble. If a nybble causes an overflow, it is reduced until the situation is rectified. Nybbles are combined into bytes and written to the output file.

The file can then be programmed into an EPROM and incorporated in the CDV-1 circuit. Make sure the circuit is set to CDPCM mode (no jumper), plug in the batteries, 8-Ω speaker, and switch, and you're ready to hear the box talk. You also need to adjust both potentiometers for proper volume and pitch.

CONCLUSION

I hope seeing how some fancy theoretical concepts can be applied to a simple, practical project has been helpful. I also meant to illustrate that some real-time signal processing tasks can be handled by a lowly PIC micro running at 1.4 MHz.

When I started working on this project, I wanted to know if it was possible to decode CVSD in firmware with an inexpensive micro. I ended up doing something entirely different and much more rewarding. Sometimes, R&D is like gambling-you follow your hunch without knowing where it will lead you. This time I hit pay dirt. ❑

Sébastien Roy holds an M.Sc. in electrical engineering from l'Université Laval, Quebec City. He specializes in embedded systems, signal processing, and practical applications of the Internet. He offers consulting and design services under the umbrella of Tertius Technologie Inc. He may be reached at roys@nbnet.nb.ca.

REFERENCES

Speech encoding:

Jayant, N.S. "Digital Coding of Speech Waveforms: PCM, DPCM, and DM Quantizers," *Proc. IEEE, 62, No. 5,* 61 1-632, 1974.

Jayant, N.S. and P. Noll. ***Digital Coding of Waveforms,*** Prentice-Hall, Englewood Cliffs, 1984.

Oliver, B.M., et al. "The Philosophy of PCM," *Proc. IEEE, 36, No.* 11, 1324-1331, 1948.

CVSD and Predictive Coding:

Greefkes, J.A. "A Digitally Companded Delta Modulation Modem for Speech Compression," Proc. IEEE ***Int. Conf. on Communications, 7.33-7.48,*** June 1970.

Markel, J.D., and A.H. Gray, Jr. ***Linear Prediction of Speech,*** Springer-Verlag, New York, 1976.

Makhoul, John. "Linear Prediction: A Tutorial Review," *Proc. IEEE, 63, No. 4, 1975.*

SOURCE

Tertius Technologie, Inc.
RR #2, Site 17, Box 9
Beresford, NB
Canada EOB 1H0
(506) 542-1618

CDV- 1 kit includes documentation, software, sample sound files, PCB, programmed PIC, other ICs, and discretes. Speaker, power supply, reset switch and EPROM not included	$29.95
CDV-1 A&T unit	$32.95
Programmed PIC	$10
Programmed EPROM	$10
PC board, no silkscreen	$12

All prices in U.S. dollars. Shipping extra. Call for OEM pricing.

IRS

404 Very Useful
405 Moderately Useful
406 Not Useful

FEATURE ARTICLE

Matt Park

Real-Time Digital Signal Processing with a PC and Sound Card

A Windows-compatible sound card and a 486 PC are all you need to experiment with real-time digital signal processing. Here, Matt introduces us to Digitize, a real-time software DSP alternative.

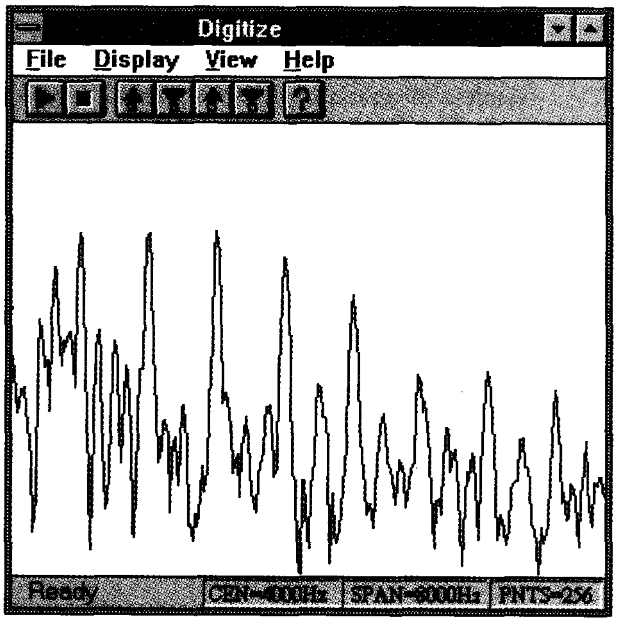
I wanted a graphic-equalizer display on my PC, but it took a friend's need to get me going. After he asked me how to display a narrow-band IF spectrum from his amateur radio receiver on his PC, I started to design and write Digitize, a Windows-based spectrum analyzer. My program uses a sound card to digitize audio frequencies and a PC to calculate and display over 40 512-point FFTs per second.

I am not suggesting that a 486 PC and standard sound card replace a dedicated DSP chip. DSP chips are significantly faster and more powerful. However, most people do not have a general-purpose DSP chip in their PC. Based on my experience with Digitize, you do not need a dedicated DSP to start writing real-time DSP programs. I also believe that today's inexpensive sound cards have practical DSP potential in embedded PCs.

In this article, Digitize serves as an example of sound-card-based, real-time digital signal processing under Windows. I'll discuss how to transfer digitized data from the sound card to a program and how to adjust recording parameters to optimize real-time performance. The techniques and code discussed in this article are applicable to all Windows-compatible sound cards. I'll also take a look at the DSP capabilities and limitations both of sound cards and Windows.

FFT AUDIO SPECTRUM DISPLAY

Digitize is a Windows 3.1 program written using VC++ 1.5. I chose Windows because its multimedia interface made programming the sound card vendor and hardware independent. I also wanted to use multitasking to calculate and display the spectrum while the PC performed other tasks. VC++ and the Microsoft Foundation Classes simplified the programming required to implement Digitize's Windows interface (see Figure 1).



Ready | CEN=4000Hz | SPAN=8000Hz | PNTS=256

Figure I--The *typical* Digitize *window offers the usual Windows options: File, View, Help, hot keys, a tool bar, and a status bar. The Display menu offers choices for each of the tool-bar buttons and a speed check routine.*

However, since VC++ uses the standard Windows Multimedia API to interface with the sound card, any Windows-compatible compiler can be used to program with the Multimedia API. Even though I used a C++ compiler, Digitize contains mostly standard C code inside the member functions of VC++.

As Table 1 demonstrates, Digitize calculates and displays FFTs extremely fast. The number of FFT points and sampling characteristics are adjustable. Using the setup depicted in Figure 2, I can calculate and plot more than 30 FFTs and plots per second while using Windows for other tasks.

Digitize has three main sections: data acquisition, digital signal processing, and data output. The data-acquisition routines interface with the sound card to digitize or record data, the DSP section calculates the FFTs, and the output section plots the spectral display. For future work, the DSP and output sections will change, but the low-level audio-input routines should require very little modification.

PC	# Data Pts	FFT&Plots/s	FFT/s	Plots/s	Video
Pentium 60*	512	132 (0.008)	558 (0.002)	217 (0.005)	accel. VGA
486/100	512	91 (0.011)	260 (0.004)	260 (0.004)	accel. VGA
486/66	512	62 (0.016)	160 (0.019)	200 (0.005)	accel. VGA
486150	512	49 (0.020)	129 (0.024)	198 (0.005)	accel. VGA
486/33	512	34 (0.029)	a2 (0.012)	1a7 (0.005)	accel. 800 x 600
486/33	512	13 (0.073)	a2 (0.012)	20 (0.050)	SVGA 800 x 600
486/33*	512	22 (0.045)	a9 (0.011)	197 (0.005)	accel. 800 x 600
Pentium 60*	1024	72 (0.014)	260 (0.004)	130 (0.008)	accel.VGA
486/50	1024	24 (0.041)	56 (0.01 a)	100 (0.010)	accel. VGA

*Tested running on NT Server 3.5

Table 1—*Typical* speed results *of* Digitize 1.0 show just how *fast it runs. A 256-point FFT uses 512 data points. The number shown in parentheses represents seconds per calculation. For example, (0.008) represents 8 ms per FFT.*

WINDOWS AUDIO-INTERFACE FUNCTIONS

Windows provides a high- and low-level method of accessing sound cards and other multimedia hardware. The high-level method is file based. Its audio functions are easy to use and useful for digitizing data to disk. However, they require non-real-time processing of the data.

To continuously access data as soon after it is digitized as possible, the DSP programmer must use Windows Multimedia low-level audio functions. Windows sends messages to inform a program when events happen. As well, Windows programs are written to respond to events or messages. Events can include mouse clicks, keystrokes, or a full buffer of digitized data.

Multimedia programs interact with the low-level audio services to transfer the digitized data from the sound card. Under Windows, programs do not directly read from and write to the sound card. Instead, the programs call and respond to the low-level audio functions and messages. The audio functions and the sound card's driver deal the sound card itself.

THE RECORDING PROCESS

The user begins the recording process by clicking the record tool-bar button, typing Ctrl-R, or selecting the Start Display menu item. All of these commands send a Windows message to the C++ message map resulting in a call to the function CDigitizeView: OnRecord() presented in Listing 1.

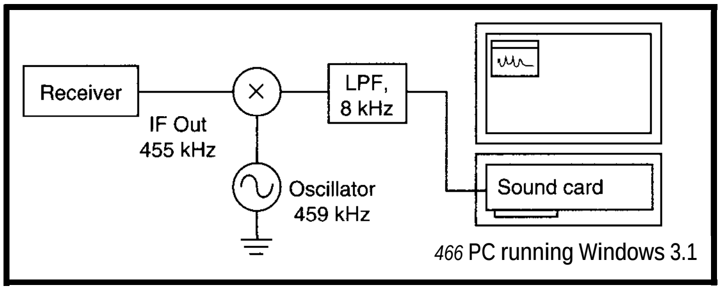


Figure 2-*Using a DSP sound board and some front-end circuitry, it's possible to calculate and plot more than 30 FFTs and plots per second while using Windows for other tasks.*

OnRecord() fills the multimedia PCMWAVEFORMAT and WAVEFORMAT structures with specific sampling details such as sample rate, bits per sample, and the number of channels to sample [stereo or mono).

WaveInOpen() iscallednext. I specifythe CALLBACK_WINDOW flag so the waveform input messages are sent to Digitize for processing. Because VC++ does not use Windows handles in the same way as the Windows 3.1 SDK,Iuse GetSafeHwnd() to dynamically determine Digitize's window handle.

The MM_WIM_OPEN message (see Listing 2) is sent to Digitize from Windows in response to waveInOpen(). At this point, the recording has not started. MM_WIM_OPEN() fills the WAVEHDR structure with the appropriate buffer size and flags. Next, waveInPrepareBuffer() preparesthe data buffer for audio input and waveInAddBuffer() sends thebufferto the sound-card driver. After calling waveInStart(), MM_WIM_DATA messages are generated whenever the buffer is filled, signaling Digitize that the buffer is ready for use.

The OnMM_WIM_DATA function of Listing 3 processes the MM_WIM_DATA messages. First, waveIn-Unprepare() unprepares the header. Digitize then again prepares and sends the data buffer (pBuffer) back to the sound card to be filled so no data is lost.

The data buffer should be filled through the DMA transferring concurrently with your signal processing. Gaps might appear in the digitized data stream if the buffer is prepared after the DSP calculations take place. A ping-pong buffer scheme is often used in DSP to minimize gaps that might occur in the data. I stopped using ping-pong buffers after determining that the spectral plots would not be affected by using a single buffer.

When the user requests Digitize to stop recording, OnRecordEnd() is called. It in turn calls waveInReset(), which results in the MM_WIM_CLOSEmessage.OnMM_WIM_CLOSEO of Listing 4 responds to the stop-recording message, frees the allocated data buffers, and takes care of other required housekeeping chores.

I placed OnMM_WIM_OPEN(), OnMM_WIM_DATAO,andOnMM_WIM_C LOSE ( ) in the VC++ message map to process the waveform audio messages. If you program in Windows SDK and C, the code inside these functions becomes part of a long switch function block.

Listing 1—On Re co rd ( ) is the first function called to begin the recording process.

```
void CDigitizeView::OnRecordO

  hBuffer = GlobalAlloc (GHND| GMEM_SHARE,
              m_Buffer_Size);  // Allocate memory

   if (hBuffer == NULL) {
     MessageBeep(MB_ICONEXCLAMATION);
     MessageBox(szMemError, szAppName, MB_ICONEXCLAMATION |MB_OK);
      return:

//pointer to allocated buffer location
  pBuffer = (LPSTR)GlobalLock (hBuffer);

// Open Waveform Audio for Input
  pcm.wf wFormatTag       = WAVE_FORMAT_PCM;
  pcm.wf nChannels        = m_Num_of_Channels;
  pcm.wf nSamplesPerSec   = m_SampleRate;
  pcm.wf nAvgBytesPerSec  = (WORD)(m_SampleRate*m_Num_of_Channels*
                              m_Bits_Per_Sample/8);
  pcm.wf nBlockAlign      = (WORD)(m_Num_of_Channels*
                              m_Bits_Per_Sample/8);
  pcm.wB tsPerSample      = m_Bits_Per_Sample;

   if (::waveInOpen((LPHWAVEIN)&hWaveIn, 0, (LPWAVEFORMAT)&pcm.wf,
       (UINT) GetSafeHwnd(), OL, (DWORD)CALLBACK_WINDOW)){
     GlobalUnlock(hBuffer);
     GlobalFree(hBuffer);
     MessageBeep(MB_ICONEXCLAMATION);
     MessageBox(szOpenError, szAppName, MB_ICONEXCLAMATION |MB_OK);
   }
   starttime = GetTickCount();  // get the starting ii of seconds
                                // for speed check
  Time-Through = 0:             // reset the counter
```

LOW-LEVEL AUDIO DATA FORMAT

The sampled data's format depends on the number of bits and channels per sample. If the sound card is set for 8-bit samples, then each sample has 1 unsigned byte (8-bit offset) with a value between 0 and 255 (with 0 V = 128).

If the number of bits per sample is 8-16 bits, then each sample is a 2-byte signed integer in the range –32768–+32767. As is standard with Intel processors, the least-significant byte is stored first. As Table 2 indicates, for stereo data, the left channel's sample precedes the right channel's sample.

FFTS AND SPEED

All the data digitized by the sound card is real data. However, FFT transforms require complex inputs and produce complex outputs. By taking advantage of the even and odd symmetry of the FFT, 2N real data points can be loaded into an N-point complex array to calculate an N-point FFT.

The first sample is loaded into the real part of the first complex array element. The second real sample goes into the imaginary part of the first complex array element, and so on. Calculating an N-point FFT filled with 2N real data points saves half the number of complex multiplies and additions with respect to a 2N FFT filled with 2N real data points and zeros for the imaginary data points.

No spectral information or resolution is lost by using this real-to-complex FFT technique. Before the FFT's output can be used, the real and imaginary output bins must be combined using an additional butterfly. A fast FFT routine is important for the Digitize's speed and for future DSP experimentation, which may use FFTs to implement fast convolution, correlation, and FIR filters.

I designed Digitize to calculate and plot FFTs as fast as possible. With the exception of a scaleable window and variable FFT point size, I purposely did not include extra bells and whistles.

Digitize uses an assembly language FFT routine. Brian McCleod optimized the Fft_R ( ) routine [1] so that it can now calculate over 125 FFTs per second for 512 real data points. McCleod believes he can further optimize the routine to exceed 200 FFTs per second. A faster 1 og ( ) routine based on a look-up table can be implemented to further improve Digitize's FFT speed.

Digitize can be set to use any power-of-two FFT size up to 8192 by changing the fft_points entry in DIGITIZE.INI.Ihavesettledona 512-point FFT (512 real data points) for pan display and general use.

All the FFT routines that I use require floating-point input values between -1 .0 and +1.0. A loop converts the sampled data to match the FFT's input, so it is a convenient place to implement time-domain windowing. For a detailed look at FFT spectral analysis and time-domain windows, refer to "Spectral Analysis: FFTs and Beyond" [2].

Digitize can be configured for the following time-domain windows: rectangular, Hamming, Hanning, and Blackman. The OnMM_WIM_DATAO function of Listing 3 shows how Digitize converts the sampled data to match the FFT's input.

Digitize was originally designed for a pan display, which displays the frequency spectrum of RF carriers and sidebands. The RF signals can be modeled as periodic signals, so plotting the square of the FFT's output magnitude produced good results. However, as Digitize became faster and more useful for analyzing noisy, nonperiodic, or random signals such as speech, I modified it to also plot the average spectral density of the signal.

The periodogram is a common method for calculating the average spectral density of a signal. Periodograms divide the samples into overlapping segments, perform an FFT on each segment, and average the results. Generally, a 50-75 % overlap provides 90% of the possible performance improvements [2].

Time-domain windows are used in periodograms to help reduce the degradation caused by transforming the finite number of data points and the sidelobes that result. Digitize has an option to overlap data segments by 50% of the FFT size [3].

Listing 2-On MM_WIM_OPEN() prepares the data buffers and calls the function to start the actual transfer of data.

```
long CDigitizeView::OnMM_WIM_OPEN(WPARAM wParam1, LPARAM lParam1)
{       // Set Up the WAVEHDR structure to be filled with data

  pWaveHdr->lpData          = pBuffer;        //already allocated
  pWaveHdr->dwBufferLength  = m_Buffer_Size;//from ini file
  pWaveHdr->dwBytesRecorded = 0L;             //valid after filled
  pWaveHdr->dwUser          = 0L;
  pWaveHdr->dwFlags         = 0L;
  pWaveHdr->dwLoops         = 1L;             //only used for output
  pWaveHdr->lpNext          = NULL:           //reserved do not use
  pWaveHdr->reserved        = 0L;             //reserved do not use

// Prepare and Add
  waveInPrepareHeader(hWaveIn, pWaveHdr, sizeof (WAVEHDR));
  waveInAddBuffer(hWaveIn, pWaveHdr, sizeof (WAVEHDR));

// Begin Sampling
  waveInStart(hWaveIn);
  return 0;
}
```

SOUND-CARD-BASED, REAL-TIME DSP PROCESSING

I have come across four limitations to full-blown DSP programming using Windows and sound cards:

- standard sound cards have only one DMA channel for transferring data and cannot record and play simultaneously
- there is a delay before the data is available for use because of the 2048-byte minimum DMA data transfer
- sound cards are not DC coupled, which limits their use for feedback and control. Sound cards' audio passbands often start around 30 Hz.
- Windows is not a preemptive multitasking operating system and certain events can slow or stop the message-processing loop, preventing the data from being accessed

Most of these limitations can be overcome to make the sound card and Windows suitable for real-time processing. It is important that you read your sound card's manual and experiment with your equipment to maximize hardware capabilities.

At the moment, the single-channel DMA limitation under Windows is difficult to overcome without buying a new sound card with two DMA channels or a DSP chip. In the future, I intend to experiment with direct reading and writing of the sound card's ADC and DAC. The ability to read and write simultaneously will be a decision factor when I purchase my next sound card [4].

The single record or playback capability limits a sound card's usefulness for feedback and control or real-time filtering applications. However, the single-channel DMA is not a limitation for applications where the PC detects, displays, or recognizes events.

For example, I can envision a laptop with an internal sound card being used by a field technician to analyze the output of an accelerometer. Such a tool could determine if a moving piece of equipment is vibrating within factory specifications. The same laptop then becomes useful for data entry and as an intelligent piece of test equipment.

Quick data access is important for applications such as a pan display, which requires minimal lag time between when the input changes and when it is recognized by the PC.

When Digitize was set for slower sampling rates, a noticeable delay occurred between a change in the receiver's demodulated audio output and the plotted spectrum. There are several ways to minimize the delay required to fill the minimum 2048-byte DMA buffer. One solution is to increase the sampling rate and fill the buffer faster. The MPC 1 .O specifications require a sound card to support the 11,025- and 22,050-Hz sampling rates. Other sampling rates are optional, although the 44.1 -kHz sample rate is commonly supported.

Experiment with your sample rate settings to determine what your card supports. Mine samples a single channel from 4001 Hz to 44.1 kHz.

8-bit Mono	0	1	2	3		2044	2045	2046	2047
8-bit Stereo	O-L	O-R	1-L	1-R	:::	1022-L	1022-R	1023-L	1023-R
16-bit Mono	0 LSB	0 MSB	1 LSB	1 MSB		1022LSB	1022 MSB	1023 LSB	1023 MSB
16-bit Stereo	O-L LSB	O-L MSB	O-R LSB	O-R MSB	...	511-L LSB	511-L MSB	511-R LSB	511-R MSB

L, R = left or right channel
0, 1, 2.. = sample numbers

Table 2—Waveform audio data is stored in differenf formats within a 2048-byte buffer depending on how it's sampled.

Other boards may support a different set of sampling rates, which may change depending on stereo or mono sampling.

Using multirate digital signal processing, the sampling rate can be increased to f_s without requiring an increase in the speed of the signal processing routine. The sample rate is reduced to f_s/N by decimating or only using the N^{th} data samples.

In addition, a combination of analog or digital antialiasing filters ensures that the signal is bandwidth limited to $f_s/2N$ prior to decimation. A special case of decimation uses the selection of stereo, mono, 8-bit, or 16-bit sampling modes to change the buffer fill rate.

In Listing 3, there are examples of how to decimate the data buffer for each of the sampling modes. A 2048-byte buffer with 16-bit stereo sampling at 11,025 Hz is filled every 46 ms. At 8-bit mono sampling, it takes four times longer (186 ms) to fill the buffer.

One advantage of using a combination of the stereo, mono, 8-bit, or

Listing 3—*OnMM_WIM_DATA()is the main* function in Digitize *which handles the sampled data. The MM_WIM_DATA message is sent whenever a data buffer is full of data.*

```
long CDigitizeView::OnMM_WIM_DATA(WPARAM wParam1, LPARAM lParam1)
{
  unsigned int i:  //counter for translate loop
  unsigned int k; //counter for fft per buffer loop

  waveInUnprepareHeader(hWaveIn, pWaveHdr, sizeof (WAVEHDR));
  pBuffer16 = (int far *)pBuffer;    //use int for 16-bit samples

// Set the Header, Prepare, and Add
  pWaveHdr->lpData          = pBuffer;
  pWaveHdr->dwBufferLength  = m_Buffer_Size;
  pWaveHdr->dwBytesRecorded = OL:     // valid after filled
  pWaveHdr->dwUser          = 0L;
  pWaveHdr->dwFlags         = 0L;
  pWaveHdr->dwLoops         = 1L;
  pWaveHdr->lpNext          = NULL:   // reserved. Do not use.
  pWaveHdr->reserved        = 0L;     // reserved. Do not use.

  waveInPrepareHeader(hWaveIn, pWaveHdr, sizeof(WAVEHDR));
  waveInAddBuffer(hWaveIn, pWaveHdr, sizeof(WAVEHDR));

//loop to handle multiple FFTs per buffer
  for (k = 0; k <m_FFTs_Per_Buffer; k++){
      // loop to translate from char to double read in _FFT_Size
      // number of data points. Adjust conversion for digitizing
      // mode 8 or 16 bits, stereo or mono. In addition to
      // converting from integers to floats the (left) samples
      // are multipleid by the DSP windowing function.
```

(continued)

16-bit sampling modes and standard sound-card sampling rates is that most sound cards automatically configure an antialiasing filter based on the sample rate. I found that the quality of the antialiasing filter depends on the manufacturer.

There is no one correct answer for every application. The goal is to minimize the time lag required to fill the data buffer and to maximize the throughput of the calculations. To do so, the combination of sample rate, buffer fill time, and decimation should result in the final desired data size being filled in just over the time it takes to calculate one complete DSP algorithm.

There is no speed penalty for sampling with 16 bits instead of 8, since both 16- and S-bit data samples are converted to floating-point numbers before being passed to the FFT calculation. For minimal data delay, it's advantageous to sample with 16 bits using only the most-significant byte if the processing routine does not require 16-bit resolution.

With my 8-bit sound card, I ended up using a combination of decimation (stereo sampling) and multiple FFTs per data buffer to minimize the data access delay and maximize the number of plots per second, even though Digitize only uses data from the left channel. I set the sampling to a stereo, 16-kHz sampling rate, two FFTs per buffer, and my PC was able to reliably operate at 30 FFTs and plots per second.

DIGITIZE AND WINDOWS

Even though Windows is not a real-time or preemptive multitasking operating system, by using the proper sampling and FFT settings, I found that my PC could update the plot window very quickly.

Nonetheless, I have found three characteristics of Windows which adversely affect real-time processing:

- pull-down menus
- extensive disk access
- activating the Task Manager

Both pull-down menus and Task Manager slow down the message-processing loop which affects data access. When Digitize was running at its maximum speed (at >40 FFTs and plots per second, Windows can still respond to mouse clicks and key strokes), activating the Task Manager or opening one of Digitize's menus for an extended period of time would sometimes freeze Windows from responding to user input, even though Digitize continued to process data and display spectrum plots.

Listing 3-continued

```
    switch (m_Bits_Per_Sample| m_Num_of_Channels){
      case EIGHT-MONO:
        for (i=0;i<m_FFT_Size;i++){ //translate from char to floal
          RealData[i]=(float) (((BYTE)pBuffer[i + (k*m_FFT_Size)]
                         127)/128.0F*mag_Weights[i]);
        }
        break:

      case EIGHT-STEREO:
        for (i=0;i<m_FFT_Size;i++){ //translate from char to float
          RealData[i]=(float)(((BYTE)pBuffer[2*(i+(k*m_FFT_Size))]-
                                    127)/128.0F*mag_Weights[i]);
        }
        break;

      case SIXTEEN_MONO:
        for (i=0;i<m_FFT_Size;i++){// translate from int to float
          RealData[i] =(float)( ((int)pBuffer16[i +
                         (k*m_FFT_Size)])/32768.0F*mag_Weights[i]);
        }
        break:

      case SIXTEEN-STEREO:
        for (i=0;i<m_FFT_Size;i++){ //translate from int to float
          RealData[i]=(float) (((int)pBuffer16[2*(i +
                        (2*k*m_FFT_Size))])/32768.0F*mag_Weights[i]);
        }
        break:
    }                //end of digitizing mode switch statement

// compute forward FFT of data
    Fft_R(RealData, ImagData, fftpower-1, 1);

// calcualate log magnitude of each FFT point and convert to int
    for (i = 0; i < m_FFT_Size/2; i++){
      mag = (double)(RealData[i]*RealData[i] +
                         ImagData[i]*ImagData[i]);
      // =1/threshold setting from ini file
      if (mag <= m_Threshold)
        mag = m_Threshold;
      mag_FFT_Points[i] = (unsigned int)
                              (m_Gain*(double)log10(mag)+m_Offset);
      //mag_FFT_Points[i] = (mag_FFT_Points[i] +
                                mag_FFT_Old[i])*10/20; //averages
    }
    //mag_FFT_Old = mag_FFT_Points;
    // graph it pass the values to graphing routine
    OnStartGraph(m_FFT_Size/2, mag_FFT_Points);

  } //end of multiple FFTs per buffer

  return TRUE;
}
```

For real-time processing, you should either ensure that your processing loop takes Windows' variable time overhead into account or you should limit the amount of time some features can be active. Digitize disables the menus when recording, however the tool bar and hot keys still work.

Listing 4—*OnMM_WIM_CLOSE()* *is the last function called to stop the recording process.*

```
long CDigitizeView::OnMM_WIM_CLOSE(WPARAM wParam1, LPARAM lParam1)
{
// Free the Buffer Memory
  GlobalUnlock (hBuffer) ;
  GlobalFree (hBuffer) ;
  bRecording = FALSE ;
  return TRUE ;
}
```

Digitize runs under Windows NT and I have considered switching from Windows 3.1 to Windows NT since NT's preemptive multitasking would overcome some current limitations and still maintain low-level audio compatibility.

SUMMARY

In the end, my friend has a higher-performance pan display than he really needed and I got a great start experimenting with DSP on my home PC.

I was surprised at the DSP potential of my PC. With the higher FFT speeds that I expect to achieve through optimization, a real-time sonogram is feasible. I plan to modify the display code to plot a sonogram with playback capability. The sonogram could then be used as part of a computer-based language tutor.

Other possible projects in real-time processing include oscilloscope displays, DTMF decoders, and an audio spectrum analyzer with a swept sine-tracking generator.

One of these days I may even find the time to modify the display portion of Digitize to plot a 16-band graphic equalizer display, which is what I wanted in the first place.

Matt Park holds a B.S.E.E from Ohio State University and a M.S.E.E from Johns Hopkins University. He is an engineering consultant in the areas of communications and electrooptics. He may be reached at jmpark@tpts1.seed.net.tw.

REFERENCES

[1] J.G.G. Dobbe, "Algorithmn Alley: Faster FFTs," *Dr. Dobbs Journal*, February 1995.

[2] Ifeachor and Jervis, "Digitial Signal Processing: A Practical Approach," Addison-Wesley, 1993.

[3] David Prutchi, "Spectral Analysis: FFTs and Beyond," *INK*, 52, 20–39, 1994.

[4] Bob Fine, "Make the Most of Your DSP-based Sound Card", *INK*, **53**, 30–38, 1994.

I R S

407 Very Useful
408 Moderately Useful
409 Not Useful

PC-based Equalizer

FEATURE ARTICLE

Eric Ambrosino

Multimedia on the PC has given new dimensions to the importance of PC sound. In this article, Eric presents an equalizer with a graphical interface comparable to what you would find on a home stereo.

multimedia and its use of sound has taken off. Yet, with the growing number of sound cards and the options available, manufacturers still exclude the necessity for graphic equalization. Multimedia users would find programmable graphic equalization curves that store as EQ files a worthwhile tool.

To meet this need, I set out to achieve programmable audio equalization along with a few benefits and options in a PC-based setting. My solution: the PC EQ project kit, a multimedia sound enhancement plug-in card for IBM PC compatibles running Windows 3.1 programs.

PC EQ provides programmable, stereo, seven-band graphic equalization with individual volume and EQ controls for left and right channels. The installation and operation is easy (it requires only an I/O base address). There's no need to worry about messy IRQs or DMA channel settings. For those who do not have a Wavetable upgrade option on their sound card, this project provides a solution.

You can program custom equalization curves for multimedia applications and benefit from wavetable synthesis as well. PC EQ, pictured in Photo 1, does this by providing a Wave-compatible 26-pin connector for adding various wavetable synthesis daughterboards.

There are several types of wavetable daughterboards readily available on the market. Owners who do not want to discard their original sound card now have an upgrade path available to them. For example, PC EQ can beef up the sound of your original card with the equalization section and accomodate various wavetable daughterboards. Even if your current sound source is wavetable based, you can add equalization and wavetable sounds for an even richer sound.

The card also features PC/104 and mezzanine analog and digital bus options. The mezzanine bus provides future options for remote or automated operation of the unit. With this option, PC EQ's presets can be switched in real time from a MIDI sequencer or other equipment such as MIDI keyboards, controllers, and guitar systems.

THE BEST OF BOTH WORLDS

Specifically, I needed to design the programmable equalizer portion of PC EQ to organize my MIDI studio mixing and recording applications. An extra equalizer in a MIDI studio setup always comes in handy. One that's

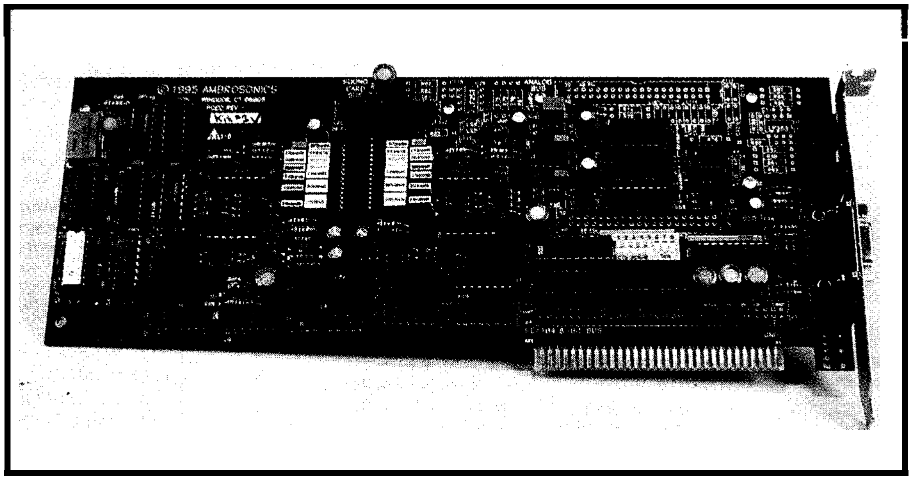


Photo I-An assembled PC *EQ* card is ready *for plug-and-play operation.*

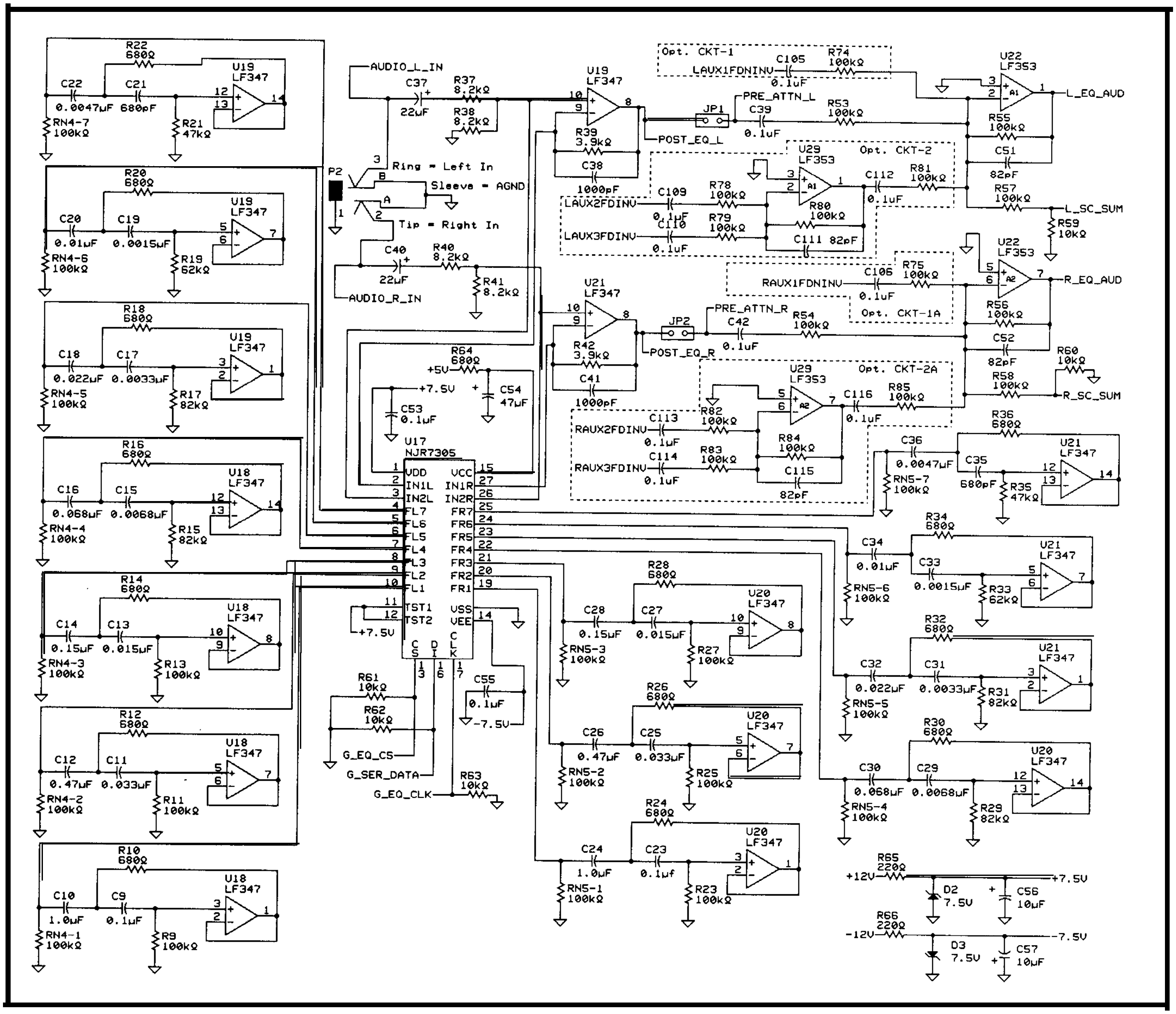


Figure 1—*At the core of the PC EQ is the New Japan Radio NJR7305 control chip. With it, true analog equalization can be accomplished under Windows-based software.*

programmable provides a new level of consistency in mixes.

I was also frustrated with being unable to upgrade to better sound with my original FM sound card. I did not want to resort to the usual "trash the old; install the new."

My goal quickly became to accommodate wavetable sound card technology as well as improve FM sound by equalization as this would give me the best of both worlds.

PC EQ'S MAIN MENU

The main menu of PC EQ is shown in Photo 2. As you can see, the controls are laid out like many graphic equalizers with the exception of a few additional controls.

Vertical scroll bars boost or cut specific frequencies by approximately ±12 dB. Seven scroll bars on each side represent the specific frequencies of the left and right stereo fields. I'll talk about these in more detail later.

Note the Dual Tracking check box in the lower midscreen area. Here, you choose whether left and right channels track in stereo or operate as two independent mono channels. Two horizontal scroll bars in the lower right control the volume level for each side and are also programmable.

The text boxes along the bottom of the screen (Preset 1-12) can be assigned unique names. When any of the 14 frequency scroll bars, check box, or volume sliders are configured to a desired setting, you can freeze the settings on the screen by saving them to any of the 12 preset text boxes. You can save various arrangements of presets and equalization curves to an EQ file specific to a CD, tape, MIDI song file, or other audio applications.

The Auto button is a feature not normally available on equalizers. By clicking Auto, a number of EQ presets, specified by the user, can be sequenced through to audition the different curves for a particular selection of music. The speed of the sequence and duration of the effect is also adjustable in a submenu within Auto.

When sequencing presets rapidly, you obtain a sort of 3D dynamic sound effect because the presets switch

Figure 2-Overall volume can *be controlled from software using a pair of programmable attenuators with optional summing amplifiers.*

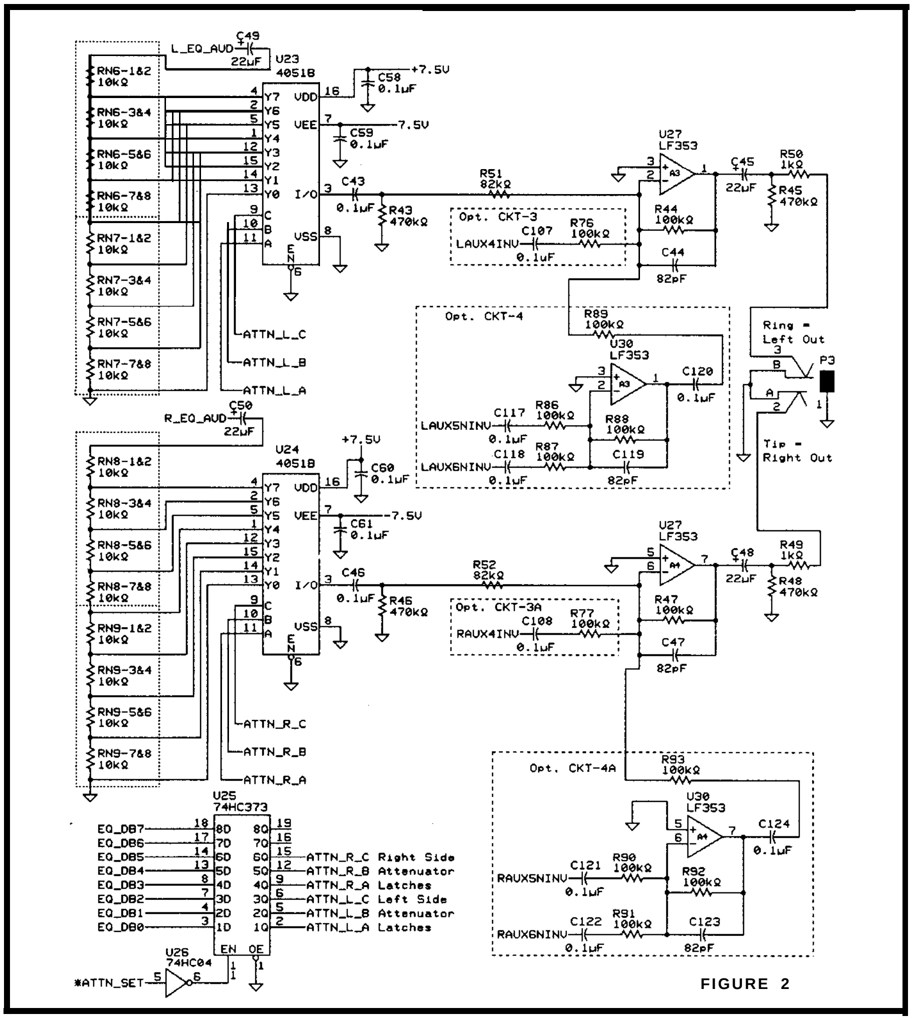


FIGURE 2

quietly. You can suspend the effect by clicking on the Stop button.

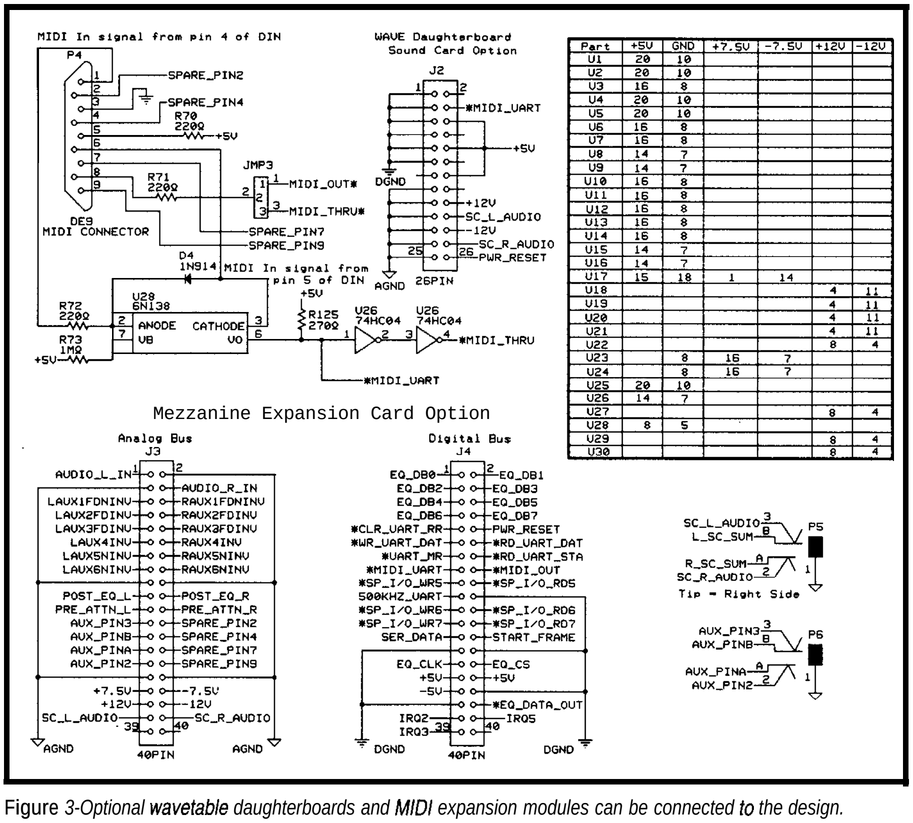
In the future, the MIDI button will offer a standard MIDI program change for selecting an EQ preset.

Finally, the menu bar along the top provides the standard file loading or saving of complete EQ files just as if you were using Windows-based word processors.

ELECTRONICS BEHIND PC EQ

Programmable equalization is accomplished by an Electrically Variable Resistor (EVR) operating in a synchronous serial scheme. The NJR7305 is is essentially 14 variable resistor networks individually addressable by an 8-bit serial data stream and synchronized to a 9-bit clock. The EVR, along with additional analog and digital circuitry and a Visual Basic program, form the heart of PC EQ.

As you can see in Figure 1, audio enters PC EQ via connector P2 and feeds the EVR (U17), which is surrounded by 14 second-order high-pass filters. Each of these filters (seven bands per channel) corresponds to a scroll bar, which is presented in the main menu of the PC EQ program. The center frequencies of each filter or scroll bar are 60, 160, and 400 Hz and 1, 2.5, 6.3, and 15 kHz.



Part	+5V	GND	+7.5V	-7.5V	+12V	-12V
U1	20	10				
U2	20	10				
U3	16	8				
U4	20	10				
U5	20	10				
U6	16	8				
U7	16	8				
U8	14	7				
U9	14	7				
U10	16	8				
U11	16	8				
U12	16	8				
U13	16	8				
U14	16	8				
U15	14	7				
U16	14	7				
U17	15	18	1	14		
U18					4	11
U19					4	11
U20					4	11
U21					4	11
U22					8	4
U23		8	16	7		
U24		8	16	7		
U25	20	10				
U26	14	7				
U27					8	4
U28	8	5				
U29					8	4
U30					8	4

Figure *3-Optional wavetable daughterboards and MIDI expansion modules can be connected to the design.*

When a scroll bar is moved, serial data is sent to the wiper of the addressed EVR channel. The associated filter sums its specific resonant frequency into the path of the input audio, which boosts or cuts that frequency band. By this technique, analog equalization is accomplished.

The frequency bands are determined by the following equations. By plugging the reference designators of the filter circuit (shown on pins 1, 2, and 3 of U18 in Figure 1) into the equations, you can custom tailor the frequency bands. You center the frequency using the equation:

Listing 1—*Visual* Basic makes writing *graphical software for the PC EQ much easier. The code here, for example, responds to equalization scroll bar changes.*

```
Sub VScrollBar_Change(Index%)
  Dim SBarValue%                   'Declare some local variables
  Dim GainCode%
  Dim Temp%

  SBarValue% = VScrollBar(Index%).Value
  AdjustValue SBarValue%, GainCode%, Index%
  GainArray(Index%) = GainCode%

 'If tracking is enabled, update the opposite side
 'Make sure ping-pong does not occur
 'when sides dont start at same point
   If Not (TrackInProgress%) Then
     If Tracking.Value = CHECKED Then    'Dual Tracking enabled
       TrackInProgress% = True           'Turn ping-pong inhibit on
       If Index% > 6 Then                'Right side adjust
         VScrollBar(Index% 7).Value = SBarValue%   'Update left
       Else 'Left side being adjusted
         VScrollBar(Index% + 7).Value = SBarValue%   'Update right
       End If
       TrackInProgress% = False          'Turn ping-pong inhibit off
     End If
   End If

 'Invoke procedure to execute I/O write to ED interface card
   If OutEnabled% Then
     SendCommand GainCode%
   End If
End Sub
```

$$f_0 = \frac{1}{2\pi\sqrt{L_0 \times C_{10}}}$$

where $L_0 = C9 \; x \; R9 \; x \; R10$. To determine filter Q and resonance characteristics, use:

$$Q_0 = \sqrt{\frac{L_0}{C10 \times R10^2}}$$

$$Q_{12\,dB} = \frac{R10 \times Q_0}{R10 + 1590}$$

Listing 1 shows the Visual Basic code that executes in response to scrolling a fader to a new position. To minimize software overhead and provide quick response to scroll bars, all faders are dimensioned in an array.

If dual tracking is on, the software writes to both stereo channels of the EVR, causing the left and right EQ channels to track one another. With dual tracking off, the channels operate independently. Vscroll Bar-Change determines which fader has been adjusted and writes the gain code corresponding to the scroll bar's new position to the selected EVR channel.

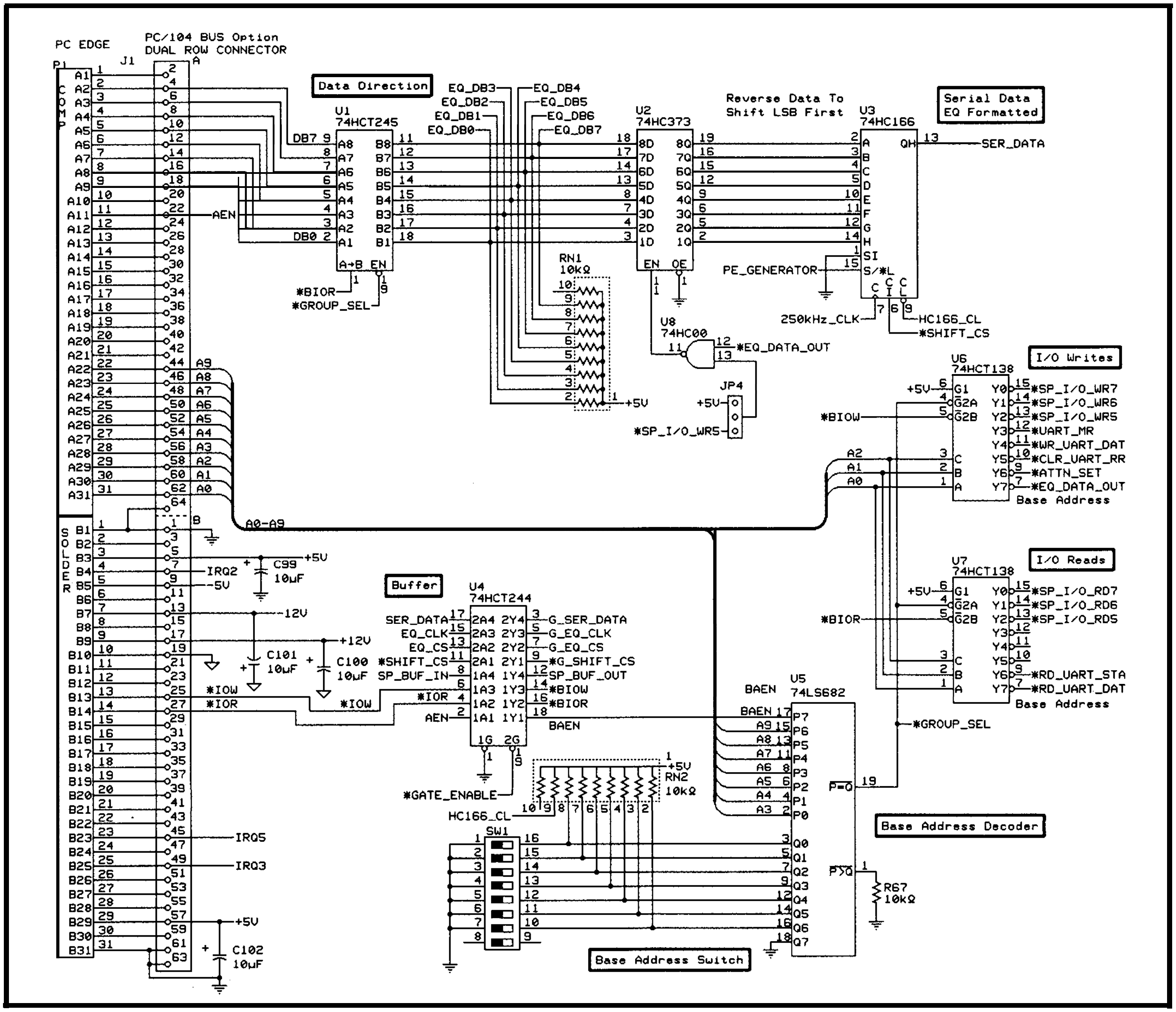


Figure 4—The decoding and interfacing logic is similar to that required on many computer plug-in cards.

The remainder of Figure 1 provides summing stages and optional auxiliary summing amplifiers used in conjunction with future EQ expansion boards. One optional board currently in the works lets you install additional EQ frequency bands. At mezzanine connectors J3 and J4, you can install optional boards. The mezzanine bus is also available if you design a card of your own. You may also populate optional circuitry and U29 to sum audio signals through the programmable stereo attenuators included on the board.

In Figure 2, left and right audio signals pass through programmable attenuators and final output stages. The final outputs are on connector P3.

Listing 2—The programmable attenuator shown in Figure 2 requires just a small amount of code.

```
Sub Attenuator_Change(Index%)
  If OutEnabled% = False Then Exit Sub

  Select Case Index%
  Case 0 'Left attenuator control
    LastAttenuation% = LastAttenuation% And BITS-012
                                        '111000 Binary value
    LastAttenuation% = Attenuator(Index%).Value Or LastAttenuation%
    agOutP Base-Address% + 1, LastAttenuation%
                                        'I/O Out (301H) to latch HC373
  Case 1 'Right attenuator control
    LastAttenuation% = LastAttenuation% And BITS-345
                                        '000111 Binary value
    LastAttenuation% = (Attenuator(Index%).Value * 8)
                          Or LastAttenuation%
    agOutP Base-Address% + 1, LastAttenuation%
                                        'I/O Out (301H) to latch HC373
  End Select
End Sub
```

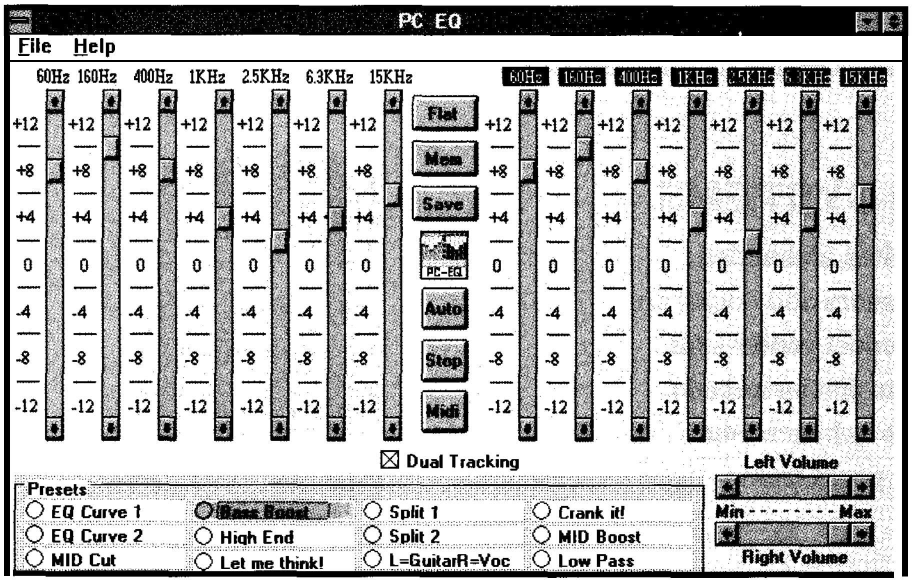


Photo 2—*The* main menu of *the* PC *EQ's* Windows program looks just like a studio or home stereo *equalizer.*

The programmable attenuators are each based on the old 4000 series CMOS 4051B multiplexer. It's a rather primitive attenuator design, but it does the job. Speed really isn't of concern because music listeners usually move an attenuator and then listen for their desired change to take effect. When running from ±7.5-V supply rails, low-level audio passing through the 4051B is primarily unaffected and, at 49¢ each, they're a bargain.

The attenuators are also dimensioned in an array and are shown in Listing 2. Data bits EQ_DB O-2 and bits 3-5 control the left and right channels, respectively. Changes made to either attenuator cause an I/O write to be initiated via the a gOutP instruction, and are latched by U25, a 74HC373.

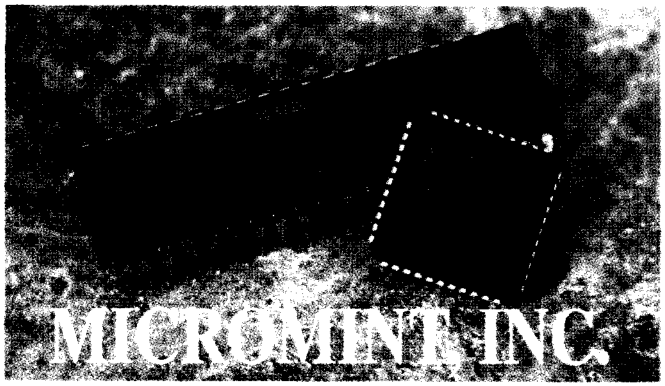
The remainder of Figure 2 supplies provisional circuitry required to reinvert signals from optional expansion boards to their proper phase.

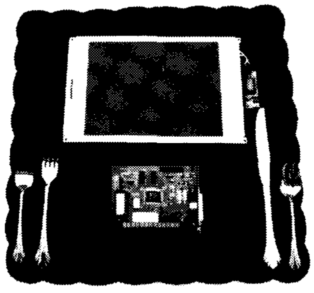
OPTIONS

I keep referring to optional circuitry, expansion boards, and mezzanine connectors J3 and J4. Let me explain how MIDI and wavetable synthesis fit into the picture.

Refer to Figure 3, and you'll notice the J3 and J4 connectors. The combination of these two connectors makes up the mezzanine expansion bus. The two connectors are located adjacent to each other on the printed circuit board. They provide a host of analog and digital signals and are meant to accept the various option boards mentioned.

MIDI In, Out, and Thru connections are provided by the 9-pin D connector, P4. The pin definition is similar to the popular Music Quest





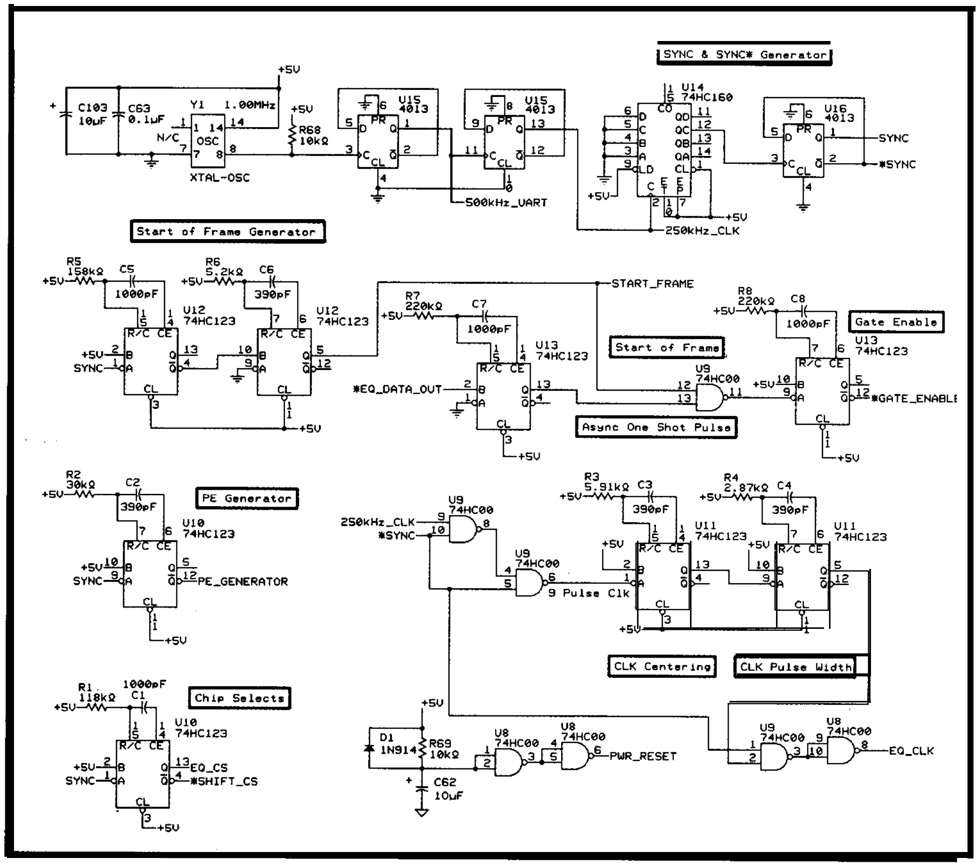


Figure 5—*The MIDI and EVR functions require* numerous clocks *and synchronization pulses.*

MQ16 MIDI card with some exceptions. The MIDI Out signal is jumper configurable as a MIDI Thru, and the spare pins are brought to J3. The J2 connector permits an optional wavetable daughterboard to attach directly to the PC EQ card. Wavetable outputs sum into the stereo attenuator section of PC EQ or are available as direct outputs at the P5 connector.

PC EQ uses some fairly common decoding logic derived from the PC's address bus and provides I/O base address selection with a DIP switch. Figure 4 illustrates the base address decoder, PC edge connector, and 8-bit PC/104 bus interface. With PC/IO4 compatibility, users have an increasing number of plug-in boards available.

For example, WinSystems PCM-Audio card is Sound Blaster compatible and is first on my list to investigate. Perhaps this option is worth considering if you haven't yet added sound to your system or your expansion slot availability is limited.

Continuing with Figure 4, PC EQ may use up to 8 I/O port addresses (U6 and U7) depending on which options are used. The PC's 8-bit data bus is directed by transceiver Ul and fed to data latch U2. The wiring between U2 and U3 is intentionally reversed, assuring that serial data from the shift register is formatted to shift out the LSB first. The serial data is now in the correct form to be clocked into the EVR circuit discussed earlier.

Finally, Figure 5 shows how the state timing is derived for the shift register, 500-kHz MIDI clock, and 9-bit EQ clock signals. Also included are the chip selects, synchronizing logic, and power-on reset circuitry.

CONCLUSION

This design offers plenty of flexibility. You can tailor your own frequency bands, add wavetable sounds, or interface with stand-alone Sound Blaster compatibility in a PC/ 104 format.

The next time you draft a letter on your word processor, pop a disc into your CD-ROM drive for some background music and launch the EQ application as well. Adjust the equalizer's scroll bars to suit your taste, and save the EQ settings as EQ files so you can play them another day.

You can get the maximum effect from your favorite PC games by postprocessing the outputs from your current sound card. You can also improve the sound characteristics from your powered bookshelf speakers by compensating for poor room acoustics.

As a whole, I've found this project to be useful and a lot of fun. It's satisfied both engineering and music needs. Now, if I could just get some recording projects finished, I'd be one happy jammer. ❑

Eric Ambrosino is an electrical engineer involved with aerospace product design. He is also a musician and designs custom electronics for all kinds of musical applications. He may be reached at Ambrosonics or at *eric.ambrosino@circellar.com.*

CONTACT

New Japan Radio (NJR)
340B E. Middlefield Rd.
Mount View, CA 94043
(415) 961-3901
Fax: (415) 969-1409

SOURCE

Ambrosonics
229 Rollingbrook
Windsor, CT 06095
(203) 688-0013

PC EQ kit containing PCB, all necessary components, executable software, and instructions $189

Dual 26-pin Wavetable Expander Module which enables comparison of wavetable daughterboards. No software is required and it only uses the PC's power supply $89

I R S

410 Very Useful
411 Moderately Useful
412 Not Useful

FEATURE ARTICLE

Paul Rubinfeld

Digital's Alpha 21164: Performance Drives Design Choices

With the Alpha chip, Digital blew open the desktop computing world. In this article, Paul specifically looks at how Alpha achieves its high performance. As with much of life, it appears that cache is the secret.

Ohe transition to desktop computing has been a challenge for Digital Equipment, but with the Alpha chip it more than made the leap. Critics, certain that Digital's days were over, have been forced to sit up and take notice.

The Alpha 21164 microprocessor is a second-generation implementation of its Alpha 64-bit RISC architecture, first introduced in February 1992. Announced in September 1994, the Alpha 2 1164 has been shipping since January 1995. Systems based on the chip were announced in April by Digital (servers) and DeskStation Technology (workstations).

This article lets you in on some of the architectural features and choices implemented to support the design goals for the Alpha chip. These goals were:

- achieve performance at least 50% higher than the 275-MHz Alpha 21064A microprocessor [Alpha 21064A was then the industry's most powerful microprocessor)
- improve performance of existing binary images to enable code optimized for earlier Alpha chips to run faster on the Alpha 21164 without recompilation
- adapt to a wide range of systems—PCs, workstations, servers, supercomputers-with different interface requirements

In general terms, the key to the Alpha 21164's performance is its four-way superscalar instruction, low latencies in functional units, high-throughput nonblocking memory subsystem with low-latency primary caches, and large second-level, on-chip write-back cache.

The design center of the Alpha 21164 is actually 300 MHz. Notably,

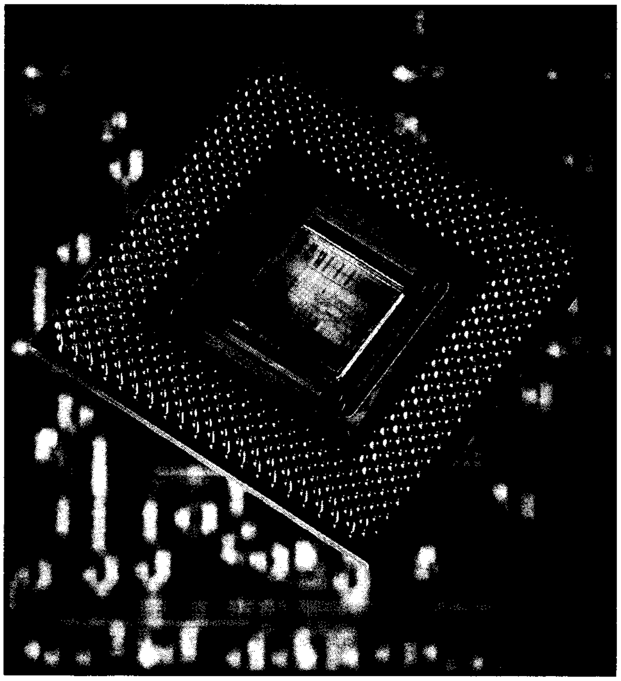


Photo I--The Alpha 21164 *boasts 9.3 million transistors,* 16.5 x *18. l-mm* die, and a design center with *a clock speed of 300 MHz.*

this measurement was taken in worst-case environmental conditions (e.g., temperature was high and voltage low). Critical paths were simulated using SPICE. The chip can be seen in Photo 1.

ALPHA 21164 MICROARCHITECTURE

The microarchitecture of the Alpha 21164 contains four basic sections: instruction unit, execution pipelines, memory unit, and bus interface (see Figure 1).

The instruction unit fetches and decodes instructions and acts as the control center for the execution units. It controls bypassing, pipeline operation, and function-unit resource allocation. It handles aborts, traps, interrupts, and exceptions, and contains the 8-KB Level 1 (L1) instruction cache. The L1 cache is organized as a direct-map virtual cache.

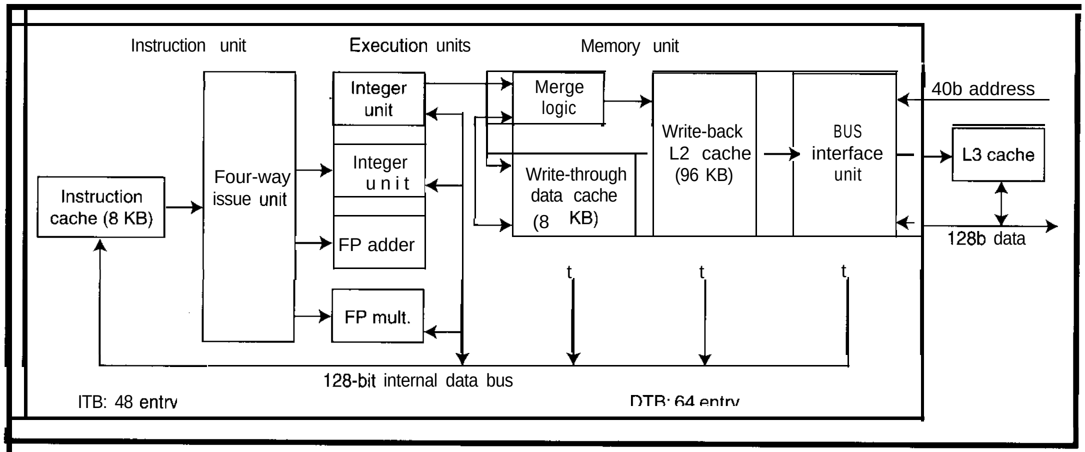


Figure 1—*The Alpha 21164 includes a number of functional units: instruction, execution (integer and floating point), memory, and bus interface unit (with cache control).*

The execution pipelines handle integer and floating-point operations. The two integer pipelines perform arithmetic and logical operations and loads. One pipeline performs shifts, stores, and integer multiplies while the other executes jumps and branches. These pipelines also generate the virtual-address calculation in the initial part of a load/store execution and operate in seven stages.

The two floating-point pipelines consist of a multiply pipeline and an add pipeline. The latter implements every operation, except multiply. These pipelines operate in nine stages.

The memory unit implements address translation and access control for data references and maintains access ordering as required by the architecture. It contains the 8-KB L1 data cache, memory load-and-store merge logic, and the Level 2 (L2) unified instruction and data cache. The L1 data cache is organized as a direct-map, write-through cache, while the L2 cache is a three-way, set-associative write-back cache.

The bus interface unit implements shared, coherent write-back caching in the large L2 cache and optional, off-chip L3 cache. It also acts as the interface to the system environment for access to main memory and I/O.

Memory data moves on 128-bit buses internally and at the pin interface.

ACHIEVING HIGH PERFORMANCE

The high performance of the Alpha 21164 is attributable primarily to three design attributes:

- high clock rate (300 MHz) with short execution latencies
- four-way superscalar instruction
- large on-chip L2 cache

From the first specification of the Alpha architecture and the design of the first Alpha microprocessor, Digital has concentrated on high clock frequency or "fast tick" as the best means of achieving high performance.

The Alpha 21164's 3.3-ns cycle-time design center was set by carefully tuning certain well-known critical and fundamental circuit paths (e.g., 64-bit adders, 64-bit shifters, and L1 data-cache access path). Once these were set, all other circuit paths were designed to operate within the cycle time.

CLOCK SPEED VERSUS COMPLEXITY

The decision to adopt a design philosophy of emphasizing clock speed over design complexity was carefully investigated for the 21164. We looked at two design approaches:

1) finding the fastest reasonable cycle time and then adding as much complexity to improve efficiency as is possible without affecting the cycle time
2) dedicating significant amounts of chip logic to improve efficiency as much as possible and then pushing the clock rate

For this purpose, we measured efficiency as the number of operations per megahertz (e.g., SPECint92/MHz). Our simulations revealed that adding complexity to improve efficiency benefits performance at a slower rate than improving the cycle time.

Recent analysis supports the validity of the Alpha 2 1164's implementation strategy. In an article in a 1993 *Microprocessor Report,* Linley Gwennap concluded that devices which emphasize high clock rates were outdistancing slower, more "efficient" designs in performance [1]. At this year's Microprocessor Forum in October, he reiterated his earlier conclusion, saying the situation favoring the "speed demons" has not changed [2].

DESIGNING A CACHE SOLUTION

The Alpha 21164 microprocessor can issue four instructions per cycle into two integer and two floating-point units for a peak execution rate of 1.2

BIPS at **300 MHz. At** this extremely high rate, the 21164 requires a tremendous number of instructions and data to approach its design performance level. The design thus requires a dual-load capability to hit its performance targets and a large, on-chip cache to minimize cache misses.

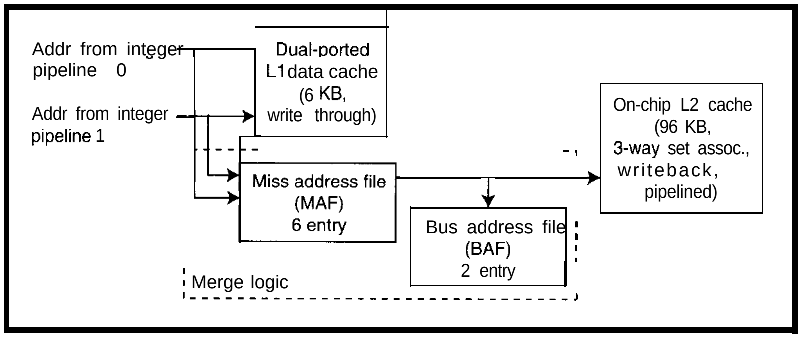

Addr from integer pipeline 0
Addr from integer pipeline 1
Dual-ported L1 data cache (6 KB, write through)
Miss address file (MAF) 6 entry
Bus address file (BAF) 2 entry
On-chip L2 cache (96 KB, 3-way set assoc., writeback, pipelined)
Merge logic

Figure **2—***The 21164 on-chip cache hierarchy offers connections between the Level 1 data cache, merge logic, and Level 2 cache.*

Conventional cache designs, combining a large L1 cache with an off-chip L2 cache, could not sustain the necessary flow. In a large array (32 KB and up), address and data propagation times exceed the 3.3-ns clock cycle.

A 16-KB L1 cache, such as in the 275-MHz Alpha 21064A, might have met the timing requirement. However, dual-porting was necessary to avoid cache banking and having to add gate delays to a critical circuit path. Increasing throughput by using a dual-ported data cache for dual-load handling would have doubled its die area-an unacceptable penalty.

This predicament led to the choice of the industry's first implementation of a two-level, on-chip cache hierarchy.

ALPHA 21164 CACHE HIERARCHY

The two-level cache hierarchy encompasses:

write-through data cache

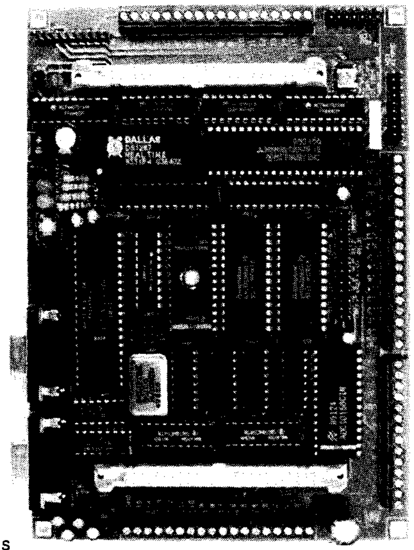
-
- merge logic

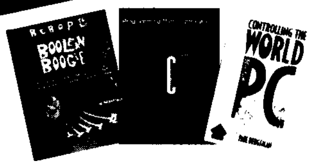
L1 write-through data cache uses 8 KB while the unified instruction and data cache is 96 KB of on-chip, three-way, set-associative write-back cache. The merge logic includes the Miss Address File (MAF) and Bus Address File (BAF). A block diagram of the cache hierarchy is shown in Figure 2.

L1 data cadual-e is read-ported, two loads to the cache can be issued with every clock cycle unless they conflict with stores in the

L1 cache misses, which are sent to the single-ported, pipelined L2 cache. The MAF optimizes load throughput by merging

32-byte block. This design enables one fill from the L2 cache to provide data for up to four merged load misses. The MAF contains six entries and queues up to 21





In the case of an L2 miss, one 64-byte block fill serves up to eight load misses. The BAF contains two entries and holds up to two L2 cache misses while the off-chip L3 cache or main system memory is probed.

The L1 and L2 designs are non-blocking (i.e., the microprocessor does not stall simply because of cache miss). A stall occurs only when an instruction is ready to issue and its data is not ready.

We did performance simulations on a very fast, small L1 cache, working in conjunction with a very large, high-throughput L2 cache. The results show a reduction in the average load latency from that of the traditional design for an enlarged, somewhat slower single-level cache hierarchy.

The two-level design enables Digital to satisfy its goal of improving performance of existing binary images. Careful tuning of the L1 cache-access path reduces cache latency by one cycle over the Alpha 21064. This reduction was possible because of its small 8-KB size which offered the same effective parallelism [roughly measured as instruction width times execution latency) in the 21164 so that it could more closely approximate the 21064.

Close matching of the 21064 enabled code tuned to first-generation Alpha microprocessors to run more efficiently on the 21164. The new microprocessor is in effect a better 21064 than the 21064 itself.

The two-level cache hierarchy also offers several significant implementation advantages. Because the L1 cache is relatively small, implementation of a truly dual-ported cache carried with it little overall die area penalty and eliminated the gate delays associated with banking the cache or stalling the pipeline because of bank conflicts. In addition, the two-level structure permitted activation of one L2 cache set at a time during accesses, saving approximately 10 W of power.

The 21164's cache hierarchy also offers the opportunity to design high-performance systems at lower cost. The L1 cache's low latency and the L2 cache's high throughput of more than 4 GBps provide excellent performance by themselves. The asymmetrical cache organization-a direct-map, write-through L1 cache backed by a three-way, set-associative writeback cache-significantly reduces the likelihood of pathological cache behavior.

As well, low-latency, high-bandwidth memory systems can be built with off-the-shelf DRAM components. At Digital, we built systems with standard page-mode DRAMs that can deliver the first 16 bytes of data for a random cache block in 96 ns, a complete 64-byte transfer every 144 ns and, for consecutive blocks in the page, a complete 64-byte transfer every 64 ns. A 1-GBps memory bandwidth is thus achieved.

Essentially, the combination of a two-level on-chip cache and closely coupled memory systems provides high-performance, uniprocessor client systems without off-chip cache. The advantages include lower system cost and simpler design.

PROGRAMMABLE PIN INTERFACE

The ability to build lower-cost, cacheless systems is due in part to the flexibility of the pin interface. For example, the switching rate for system interface communications on the Alpha 21164 pins is programmable. It can run at speeds that are fractions of the chip's high internal clock rate. This eases the interfacing task considerably.

To simplify system designs further, especially in multiprocessor servers where the memory system cannot be closely coupled to the CPU, the Alpha 2 1164 controls access to the optional off-chip L3 cache over the index bus.

The L3 cache is a direct-map writeback superset of the on-chip L2 cache. All cache policies are controlled by the microprocessor, and SRAM timing is programmable. Both synchronous and asynchronous SRAMs are supported, and programmable wave-pipelining can be used with asynchronous SRAMs.

Again, a sustained memory data rate of more than 1 GBps can be achieved. Up to two load operations with possible dirty block replacement can be in progress at a time.

This flexibility enables system designers to tune systems to their choice of technology and adapt quickly to advancing technology in memory and other system components.

CONCLUSION

The design of the Alpha 21164 chip satisfies Digital's primary goals for a second-generation, 64-bit RISC microprocessor:

. continued performance leadership
. binary compatibility with existing images at significant performance improvement
- adaptability to a range of performance-focused uniprocessor and multiprocessor system designs

A design emphasis on short cycle time over circuit complexity, an innovative, two-level, on-chip cache hierarchy, and Digital's advanced CMOS-5 process technology have enabled the company to maintain competitive advantages in both performance and time-to-market.

Sampling of the 21164 began in October 1994 and volume production was scheduled for the first quarter of 1995. Announced competitive products lag behind the 21164 in both performance and production status, either having just achieved or having yet to achieve first-pass silicon. Given the historical technology trends of the computer industry-a doubling of performance every 15 to 18 months-I believe that Digital's Alpha microprocessors hold and can maintain a 9-to-24-month lead in performance. ❑

Paul Rubinfeld is the engineering manager on the Alpha 21164 project at Digital Semiconductor, a Digital Equipment Corporation business. Paul has worked on VAX and PDP11 CPU development projects and an SIMD massively parallel processing system at Digital over the last 15 years, Paul may be reached at pir@afl.hlo.dec.com.

REFERENCES

[1] Linley Gwennap, "Speed Kills? Not for RISC Processors," ***Microprocessor Report, 7, No.*** 3, 3, ***1993.***

[2] Linley Gwennap, "Comparing RISC Processors," Talk presented at Microprocessor Forum, October 1994.

CONTACT

Digital Semiconductor
75 Reed Rd.
Hudson, MA 0 1749
(800) DEC.271 7
(508) 568-6868
Fax: (508) 568-6447

I R S

413 Very Useful
414 Moderately Useful
415 Not Useful

ALPHA 21164 SPECIFICATIONS

Process:	0.50-micron, 4-layer metal CMOS
Transistors:	9.3 million
Die size:	16.5 x 18.1 mm
Power:	3.3-v supply
Interfaces to:	5-V logic
Clock speeds:	300 MHz and 266 MHz
Maximum execution rate:	1.2 BIPS
Performance:	
Alpha 21164-300MHz =	341 SPECint92
	5 13 SPECfp92
Alpha 21164-266MHz =	302 SPECint92
	452 SPECfp92

Instruction issue:
4-way-issue superscalar (2 integer and 2 floating-point instructions per clock cycle)

On-chip cache memory:
- 16-KB (8 KB.data, 8 KB instruction) Level 1 short latency, write-through caches
- 96-KB Level 2 unified, data/instruction write-back cache

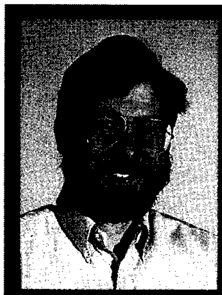
High-performance interface:
- 128-bit memory data path and 40-bit physical address
- Controller for optional, off-chip Level 3 cache
- Programmable timing, block size, and cache speed
- Interfaces to standard CMOS components.

DEPARTMENTS

FIRMWARE FURNACE

Ed Nisley

Journey to the Protected Land: Of Characters and Keystrokes



Continuing last month's keyboard discussion, Ed sets out to convert Scan Code Set 3 codes into those expected by the real-mode BIOS. He also looks at how the numerous shift keys are handled.

As you saw last month, Scan Code Set 3 reduces the keyboard interface to the falling-off-a-log level. If your brain works like mine, though, you'd rather not learn a whole new set of keyboard scan codes. The solution is, naturally enough, a simple matter of firmware.

This month, we'll convert Set 3 scan codes into the system scan codes used by the real-mode BIOS and, in the bargain, learn a lot about handling those pesky shift keys. Even though the FFTS keyboard interface doesn't precisely mimic all of the BIOS's peculiarities, the end result is familiar enough.

You can do this trick in real mode, too, but that's a whole 'nother subject!

FIELDING THE INTERRUPT

It's probably worth reviewing how we got here.

After you reset the PC, the BIOS performs its usual power-on testing and puts the keyboard into the default Scan Code Set 2 mode. Our PMLoader program disables all interrupts before switching to 32-bit protected mode and passing control to the FFTS setup code.

The code last month tested the system keyboard controller and the keyboard, then sent the commands and data required to activate Scan

Listing 1—*This interrupt handler responds to each IRQ 1 from the system keyboard controller. Pressing a key sends its single-byte make code to the controller and releasing it sends an F0 byte followed by the key's make code. A simple ring buffer holds the make and break codes until the main keyboard routine can process them.*

```
        UDATASEG
BreakCode   DD ?                        ; holds FO 00 when break found

        CODESEG

        PROC    KeyHandler
        USES    EAX,EDX,DS               ; we can't use automatic restores

        MOV     EAX,GDT_DATA            ; get addressability to our data
        MOV     DS,AX

        IN      AL,KEY_DATA             ; read scan code from controller
        MOVZX   EAX,AL                  ; clear high bytes
        Punt

        CMP     AL,0F0h                 ; are we getting a break code?
        JNE     @@Make
        XCHG    AH,AL                   ; yes, set up F0 00
        MOV     [BreakCode],EAX         ; . . . for next time
        JMP     @Done                   ; and bail out

@Make:
        CMP     [SRCount],RING_SIZE; room for one more?
        JB      @@Insert                ;  yes, tuck it away
        MOV     [BreakCode],0           ; no, flush break
        JMP     @@Done                  ; and discard it

@Insert:
        MOV     EDX,[SRHead]            ; aim at ring head
        OR      EAX,[BreakCode]         ; combine with break
        MOV     [BreakCode],0           ;  ... clear reminder
        MOV     [ScanRing + EDX*4],EAX ; save the code
        INC     [SRCount]               ; account for it
        INC     EDX                     ; tick and wrap index
        CMP     EDX,RING_SIZE
        JB      @@NoWrap
        XOR     EDX,EDX
@@NoWrap:
        MOV     [SRHead],EDX

@@Done:
        MOV     AL,NS_EOI               ; now reset the 8259 ISR
        OUT     I8259A,AL               ; EOI to primary controller

        POP     DS                      ; IRET bypasses the automatic pops
        POP     EDX
        POP     EAX

        IRET                            ; return to interrupted code

        ENDP    KeyHandler
```

Code Set 3. Some of the keyboards I tested had different Set 3 implementations, differing mostly in which keys were typematic and which were make-break. The set-up code reprogrammed the keys in the hope that all the keyboards would then work alike.

The set-up code aimed IRQ 1 at the 32-bit PM interrupt handler shown in Listing 1. The keyboard controller activates IRQ 1 when it has received and processed a byte from the keyboard. All we need do is read a single input port and process the code.

You'll recall that Scan Code Set 3 produces a single-byte make code when each key is active. Typematic keys repeat that make code at a fixed rate. Both typematic and make-break keys produce a two-byte break code when they're released: FO followed by the key's make code. Make-only keys, as you might expect, do not produce a break code.

When the keyboard is in Scan Code Set 2, the codes produced for each key depend on the state of the shift keys. Operation in Scan Code Set 3, on the other hand, produces a unique code for each key regardless of the shift state. It's up to the FFTS keyboard handler to figure out how the shifts affect the key.

The IRQ 1 handler in Listing 1 places scan codes from the keyboard controller into a ring buffer. When it reads an FO code from the controller, it sets a flag indicating that a break code is in progress and exits without changing the buffer.

When the next scan code arrives, the handler adds an FO flag to the high byte and tucks it into the ring buffer. In effect, the ring entries are just a "parallelized" version of the one- or two-byte keyboard codes.

The keyboard always sends both bytes of a break code in quick succession and, if left alone, never interposes anything else between the two. The Official IBM Enhanced Keyboard doc says that you may send commands to the keyboard at any time. That certainly implies that you may interrupt a break sequence with a command.

The real-mode BIOS tracks the keyboard state, presumably to avoid stepping on lengthy scan code sequences. The BIOS keyboard data word at 0040:0096 has several interesting flags that you should ponder before doing any real-mode surgery, particularly in Scan Code Set 2.

I simply ignored the problem, as I couldn't think of a good way to test the ensuing code. You may want to draw some state and timing diagrams when you build your interface. My guess is that many keyboards will do the right thing and, as always, others will fail at the most inopportune time. Program defensively!

There is another trap lying in wait for you assembly-language folks. The USES EAX, EDX, DS directive creates a

Listing 2—*Tasks call this routine to retrieve keystrokes from the ring buffer. The* **Key** `DoSpecials` *and* `KeyMakeChar` *routines translate raw Scan Code Set 3 values info their real-mode BIOS equivalents.*

```
        CODESEG

        PROC    KeyGtGetKey  FAR
        USES    EDX,DS,FS

        MOV     EAX,GDT_DATA
        MOV     DS,AX

        MOV     EAX,GDT_CONST
        MOV     FS,AX

        MOV     EAX,[KeyEnable]    ; are we enabled?
        CMP     EAX,O
        JE      @Done              ; nope. return nothing

;--- extract a key or die trying
@@NextKey:
        MOV     EAX,[SRCount]      ; fetch current counter
        CMP     EAX ,0             ; if zero,
        JE      @@Done             ; we are done!

        MOV     EDX,[SRTail]        aim at oldest entry
        MOV     EAX,[ScanRing  +  EDX*4];... fetch it

        DEC     [SRCount]          ; account for it
        INC     EDX                ; tick and wrap index
        CMP     EDX,RING_SIZE
        JB      @@NoWrap
        XOR     EDX,EDX
@@NoWrap:
        MOV     [SRTail],EDX

;--- process special keys
        CALL    KeyDoSpecials      ; process special keys
        CMP     EAX ,0             ; anything to return?
        JE      @@NextKey          ; no, so get another one

 --- process shift states
        CALL    KeyMakeChar        ; convert into a character
        CMP     EAX,O              ; anything to return?
        JE      @@NextKey          ; no, so get another one

@@Done:
        RET

        ENDP    KeyGtGetKey
```

few lines of code to push those registers onto the stack. In normal routines ending with a RET instruction, the assembler generates a short epilog that restores the registers before the actual RET. This is quite convenient because you only need to enter the registers in one spot and you won't get mismatched pushes and pops.

Interrupt handlers, however, must end with an IRET. That small difference throws the assembler off track: it doesn't produce the epilog. You must manually restore the registers saved by the USES directive. Nope, this isn't documented anywhere I could find. Nothing is ever simple, is it?

READING CODES

Application programs (if I may so glorify the taskettes) extract keystrokes from the buffer through the KeyGtGetKey call gate. Because FFTS is a cooperative multitasking operating system (well, sort of), KeyGtGetKey returns either a character code or a binary zero when a key isn't available.

Listing 3—*The FFTS keyboard interface returns values compatible with real-mode BIOS key scan codes and characters in AH and AL. This record shows those two bytes along with the 16 shift state bits that fill the remainder of the 32-bit EAX register.*

```
RECORD KEY-SHIFTS {
KEY_SH_SYSREQ_DN:1,       ; 15 Sys Req is pressed
KEY_SH_CAPSLOCK_DN:1,     ; 14 Caps Lock is pressed
KEY_SH_NUMLOCK_DN:1,      ; 13 Num Lock is pressed
KEY_SH_SCROLLLOCK_DN:1,   ;12 Scroll Lock is pressed

KEY_SH_ALTRIGHT_DN:1,     ;11 Right Alt is presssed
KEY_SH_CTRLRIGHT_DN:1,;10 Right Ctrl is pressed
KEY_SH_ALTLEFT_DN:1,      ; 9 Left Alt is pressed
KEY_SH_CTRLLEFT_DN:1,     ; 8 Left Ctrl is pressed

KEY_SH_INSERT:1,          ; 7 Insert mode
KEY_SH_CAPSLOCK:1,        ;6 Caps Lock
KEY_SH_NUMLOCK:1,         ; 5 Num Lock
KEY_SH_SCROLLLOCK:1,      ; 4 Scroll Lock

KEY_SH_ALT_DN:1,          ; 3 either Alt key is pressed
KEY_SH_CTRL_DN:1,         ; 2 either Ctrl key is pressed
KEY_SH_SHIFTLEFT_DN:1,    ;  1 Left Shift key is pressed
KEY_SH_SHIFTRIGHT_DN:1,   ;  0 Right Shift key is pressed

KEY_SCANCODE:8,           ; key scan code
KEY_CHARACTER:8           ; processed character
}
```

Listing *4-The three keyboard LEDs must be updated when any one of them changes. This routine maps the shift state info the LED bit locations and then sends the byfe to the keyboard. The command sequence is fair/y lengthy, so the code sends the new bits on/y if they're different from the previous value stored in* `LastLEDs`.

```
      PROC  KeyUpdateLEDs
      USES  EAX,EBX

      XOR   EBX,EBX
      TEST  [ShiftState],MASK KEY_SH_CAPSLOCK
      JZ    @@NoCaps
      OR    BL,LED_CAPS
@@NoCaps:
      TEST  [ShiftState],MASK KEY_SH_NUMLOCK
      JZ    @@NoNum
      OR    BL,LED_NUM
@@NoNum:
      TEST [ShiftState],MASK  KEY_SH_SCROLLLOCK
      JZ    @@NoScroll
      OR    BL,LED_SCROLL
@@NoScroll:
      CMP   EBX,[LastLEDs]        ; any change?
      JE    @Done

      MOV   [LastLEDs],EBX        ; yes, save new state

      CALL  KeySendCmdData,CCMD_WRCMD,CMD_NOINTS

      CALL  KeySendDataAck,KCMD_WRLEDS
      CALL  KeySendDataAck,EBX

      CALL  KeySendCmdData,CCMD_WRCMD,CMD_NORMAL

@@Done:
      RET
      ENDP  KeyUpdateLEDs
```

a)

Name	DE-25 oin	DE-9 pin	Board Header	DE-9 pin	DB-25 pin	Name
CD	8	1	1 2	6	6	DSR
RD	3	2	3 4	7	4	RTS
TD	2	3	5 6	8	5	CTS
DTR	20	4	7 8	9	22	RI
Gnd	7	5	9 10	n/c	n/c	n/c

b)

Name	DB-25 pin	DE-9 pin	Board Header	DE-9 pin	DB-25 pin	Name
CD	8		1 2	2	3	RD
TD	2	3	3 4	4	20	DTR
Gnd	7	5	5 6	6	6	DSR
RTS	4	7	7 8	8	5	CTS
RI	22	9	9 10	n/c	n/c	n/c

Table *I-The* ***PC's serial port*** *connector* ***pinouts*** *are standardized. Af* ***the*** *other end of* ***the*** *ribbon cable, however, there are* ***two*** *different 2 x 5 pin header configurations. This figure gives a fop view of* ***the*** *headers and the corresponding connector pins. The difference becomes obvious when you compare* ***the*** *two DE-9 layouts.*

It does not stall while waiting for a keystroke, lest the whole system stall with it.

KeyGtGetKey, shown in Listing 2, extracts the next entry from the ring buffer and sends it to two routines for further processing. KeyDoSpecials handles shift, lock, and other oddball keys. KeyMakeChar translates the other keys from Scan Code Set 3 into their real-mode BIOS equivalents.

As a result, KeyGtGetKey returns key scan codes that have nothing whatsoever to do with Scan Code Set 3. Even though I know this, even though I wrote the code, I still stumble while looking at the screen because I expect Scan Code Set 3 values. The upside, once you get used to it, is that all your standard PC reference books remain applicable: there's nothing new to learn about characters and scan codes.

KeyGtGetKey does not precisely mimic the real-mode BIOS response to each and every possible keystroke sequence. In particular, key combinations such as Ctrl-Break, Pause, Shift-PrtSc, Alt-SysReq, and Ctrl-Alt-Del don't behave as you'd expect. Adding some features is just a simple matter of software, others are impractical, and still others are political. As an example of the latter, I decided early on that Ctrl-Alt-Del just wasn't going to reboot the system!

That cavalier attitude would be catastrophic in a commercial operating system running standard DOS apps. There is no PC feature so insignificant that some program won't misbehave if the program is not exactly right. The PC Compatibility Barnacles are very, very solid.

FFTS is an entirely different kettle of fish. In fact, there is no reason why KeyGtGetKey should return real-mode BIOS values instead of sushi shift bits and simmered scan codes. I figured it would be easier to use it this way, but you and your application may have different requirements.

Fortunately, we have control over every program that will ever run with FFTS and can easily adapt to any self-inflicted peculiarities. At worst, we can rewrite the code to make the answer come out right.

The details lie in two routines that process each keystroke: KeyDoSpecialsandKeyMakeChar. If you need some changes, that's where you begin twiddling. Let's begin by examining what must be done.

SPECIAL ORDERS

When you press the Q key, you pretty much know what should happen: a "Q" should appear on the screen. It's not quite that intuitive, however, because you'll actually see a lowercase "q" unless you also press a shift key at the same time. If you do any typing at all, that distinction is buried in your muscle memory and you may have trouble remembering the two keystrokes separately.

The Ctrl, Shift, and Alt keys are collectively known as shift keys because they modify the result of a keystroke. The Enhanced keyboard has two of each shift key, although we expect the same result from either one of the pair. All shift keys work the same way: you must hold them down while pressing another key.

The keyboard also sports three locking shift keys: Caps Lock, Num Lock, and Scroll Lock. Because the actions associated with these keys toggle on and off, you don't have to hold them down. Each has a keyboard LED. But, contrary to some PC mythology, the LEDs aren't linked directly to the keys. We'll see how to drive the LEDs later.

The Insert key behaves like a locking shift key, although it doesn't affect the characters returned by the keyboard interface. Instead, the program using the keystrokes either replaces existing text with new characters or inserts them between the old characters. Unfortunately, there's no Insert LED on the keyboard.

It's important to realize that all the keys are the same, at least in Scan Code Set 3. Regardless of the keycap legend, each key produces a one-byte make code and a two-byte break code. How the PC interprets the codes is entirely up the program. If, for example, you'd like to have the Z key behave like the Caps Lock key and vice versa, it's a simple matter of software.

In most situations, however, it's a Good Idea to stay reasonably close to the PC standard. Once you see how it's done, though, you can twiddle the keyboard layout to suit yourself. Dvorak keyboard fans take note: it's just a few table entries away!

Ordinary real-mode BIOS functions return the scan code in AH and the character in AL. That leaves half of the 32-bit EAX register unused, which seemed a shame to me. I recycled another part of PC history by having KeyGtGetKey return the familiar BIOS shift-state bits in the high 16 bits of the register. Listing 3 shows the structure defining the bit layout.

Each locking shift key has two bits in that structure. One bit tells you the shift state: 0 for inactive and 1 for active. The other bit tells you if the key is currently pressed. In most cases, you use only the first bit, but the second makes detecting oddball chords like Alt-NumLock-Insert-M a snap.

The Shift, Ctrl, and Alt keys have one bit apiece. Those six bits tell you when the corresponding key is pressed. In addition, Ctrl and Alt each have a summary bit that is active when either key is pressed. Most of the time, you use the summary bit, but it's easy to detect Left-Ctrl-Right-Shift-Q if you must.

The SysReq key is a bit of an oddball. The real-mode BIOS detects the make and break codes, then issues Int 15, AH=85 with AL=00 and 01, respectively. The FFTS keyboard interface simply flips the bit shown in Listing 3, although you could add whatever code you feel is appropriate for your setup.

The global variable ShiftState maintains the current state of the bits showninListing3. KeyDoSpecials detects keys that change the bits and updates them appropriately. Remember that the FFTS keyboard code calls KeyDoSpecials when each key is used, so the shift states track the most recent scan code removed from the ring buffer.

The keyboard LEDs should, of course, track the state of the locking shift keys. KeyUpdateLEDs in Listing 4 converts the three shift-state bits into the corresponding LED control bits, then sends the result to the keyboard. The command sequence is lengthy enough that I decided to cache the last LED command in a global variable and update the LEDs only when they change.

Now that you know how the shift keys work, we can explore the character keys.

SHIFTING STATES

KeyMakeChar is a truly hideous routine that boils down to a single, one-line table-look-up instruction. I'll explain it without listing it here because you can't tell what's going on just from looking at the code.

Listing 2 in the June column presents KeyCodes, the table holding all the information we need to convert Scan Code Set 3 characters into real-mode BIOS values. Basically, KeyMakeChar examines the shift bits to decide which column of that table is applicable, then uses the key's scan code as an index into the table. It sounds pretty straightforward.

Deciphering the shift states, however, is a Boolean nightmare.

The first column of KeyCodes produces what IBM calls ***base case,*** when all shift and lock keys are inactive. The remaining three columns hold the values when Shift, Ctrl, and Alt are active. As you might expect, the left and right keys in each pair produce the same result.

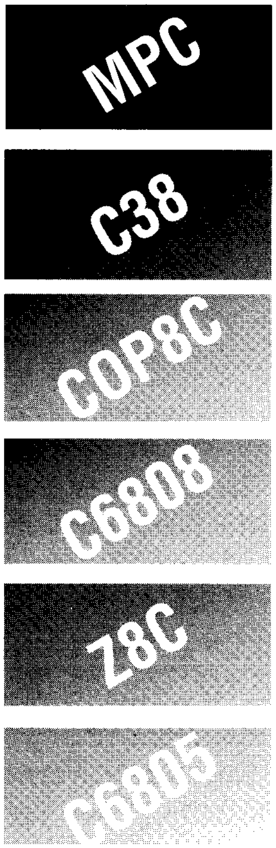
The BIOS prioritizes the various shift keys so that you can press any combination at once with any (or all] of the locking keys active and still get a character. Alt outranks Ctrl, which outranks Shift, all of which outrank the base case. If you press Ctrl-Alt-A, for example, you get the character code for Alt-A with the appropriate Ctrl and Alt bits set in the shift state.

FFTS takes a simpler tack, at least for the time being. For each character key, you may have only one shift key down in addition to any of the locking key states. This eliminates the multiply-shifted chords that come in handy at times. It also eliminates some fairly messy code that's best left until we actually need it and can do the debugging without too much extra effort.

For example, when Num Lock is active, the numeric keypad produces numbers if neither Shift key is active. Pressing Shift produces the cursor control key codes found on the Original PC keyboard, although it's easier to use the dedicated keys.

When Caps Lock is active, the letter keys produce capitals and all the other keys are unaffected. In this situation, Shift produces lowercase letters and upper-shift characters for everything else.

There are a few other twists and turns in the FFTS keyboard handler, but that should convince you that a perfect PC emulation is far from trivial. Download the code and see how your keyboard responds. The debugging information coming out the



serial port should give you plenty of insight into how things work.

HARDWARE REVISIONS

When I started on this topic, 1 found the keyboard interface on my '386SX had gone slightly sour. Given all the unsafe computing 1 do, it's a wonder the board lasted more than two years. On the other hand, not noticing a flaky keyboard for perhaps a year tells you something about the FFTS code we've been running!

1 replaced it with a 80486DX2/80 DreamTech, the folks who sold me the original '386SX. It dropped right in place with only a minor amount of twiddling. The new CPU clocks about 2400 task switches per second, roughly six times faster than the old board.

Yes, that does seem low for a 486-versus-386, dual caches versus no caches, nearly three-times clock ratio, double-width memory upgrade. I must write a column or two on performance one of these days.. As near as 1 can tell, don't believe the hype. As always, one careful measurement is worth 1,000 expert opinions.

FFTS worked perfectly after 1 tweaked a pair of delay loops to match the new CPU's speed. Shazam, all that oddball peripheral gear was up and running. Hooray for the PC Compatibility Barnacles!

I also bought a new combination 16C550 UARTs. Can you diagram the two different pinouts for those ubiquitous 2 x 5 serial-port headers? If not, tuck Table 1 in your clip-and-save file. The new l/O board was, of course, different from the old one. I spent a pleasant afternoon tracking this down and making up a set of cables to match my screwball back-panel layout.

RELEASE NOTES

There's no new code this month, oddly enough, as last month's files had the complete, working keyboard interface.

In case you've wondered what it takes to build something like FFTS, here's the box score. All told, FFTS works out to 7,200 nonblank, noncomment assembler source lines. That's about 12,000 lines or 423,000 bytes of source code distributed in 41 files. In addition, each column has some disposable demo code, specialized boot sectors, PM loaders, and so forth and so on.

Not bad for a one-man, part-time effort, eh?

Next month, we lay the groundwork for Virtual-86 mode: running "real mode" 16-bit programs in 32-bit protected mode. ❑

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.*

IRS

416 Very Useful
417 Moderately Useful
418 Not Useful

FROM THE BENCH

Jeff Bachiochi

Decontaminating the Atmosphere

FCC testing may strike fear in the hearts of engineers, but going in armed with the knowledge that you've done your own pretesting can be a comfort. Jeff checks out some inexpensive EMI probes that can help.

I'm a wee bit late for work this morning. The sticker on my windshield expires soon. I need to make a short detour through the DMV emissions bay.

This biannual test helps fight air pollution by limiting hydrocarbons and CO. I can't help but feel that if this country spent as much on research as on monitoring, we wouldn't need the monitoring. But, that might harm the petroleum industry, and we can't have that now, can we?

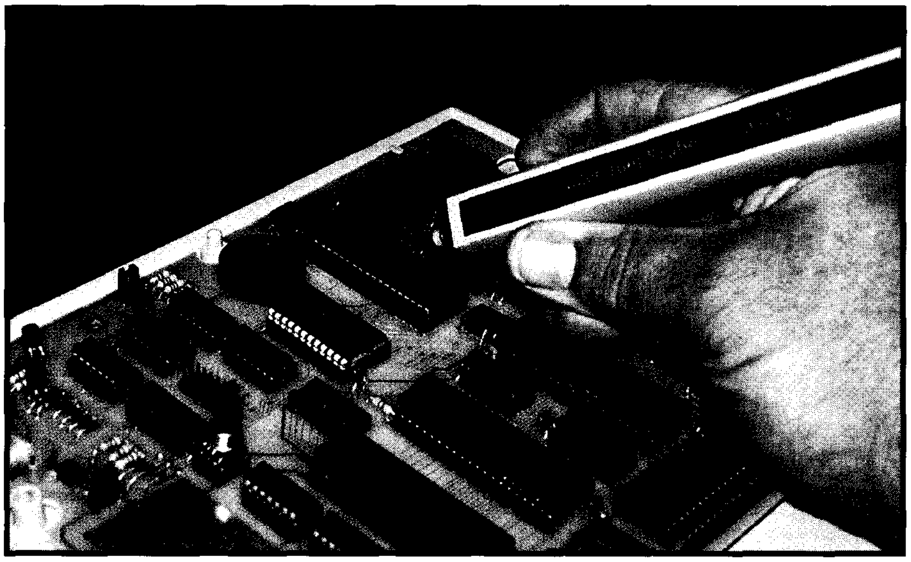
I feel a strange high as I exit the facility with a new Passed sticker on my windshield. The windows go down. I'm doing my part to keep the ozone hole from expanding. Almost immediately, an 18-wheeler passes me blowing black exhaust into my environment. I imagine its diesel power plant droning the catch phrase "Exemption." My windows go up. Now I've trapped the nasties in with me. The windows go down. Cough, coughh, coughhh.. .back to reality.

Why can't we get away from pollution? Because we invented it. After all, one person's technology is another's pollution. Who will protect us?

In the U.S., the FCC is the air police. I don't mean air as in what we breathe or the quality of program transmitted through airwaves. Instead, air is the medium supporting electromagnetic transmissions.

TV is responsible for giving us Part 15, Subpart J. That's the kicker which has us engineers jumping through hoops. Products must comply with rigid standards that protect the viewing pleasure of couch potatoes. It was John Q. Public's outcry of "interference" from our newly hatched computers that started this whole mess. Wavy lines or not, computers are here to stay. We just have to keep them contained-so to speak.

Agencies similar to FCC operate world-wide to ensure that only products which comply with electromagnetic compatibility are manufactured and sold.



Photol- When *used in* conjunction *with an oscilloscope, the Spectrum probe shows relative* *EMI* *over a wide frequency range in both the near and far fields.*

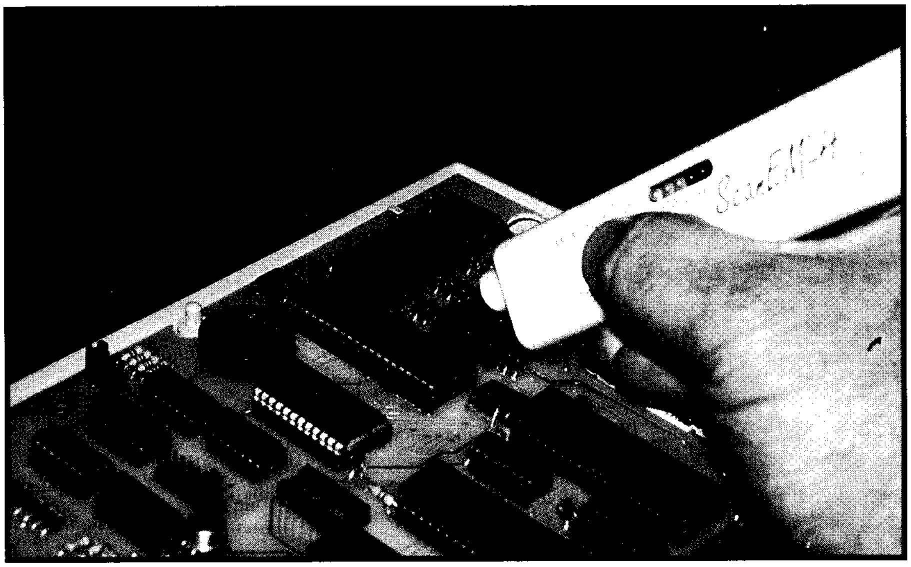


Photo P-Changes in *EMI* signal *strength for E or H fields can be seen (and heard) using the* ScanEm *hand-held* probes.

A computing device is any electronic device which generates or uses timing signals at clock rates above 10 kHz. Yup, 10 kHz—that includes your digital watch! (Transmitters and receivers are excluded because they are regulated under Part 18). These computing devices are broken down into two groups, Class A and Class B, depending on their use and marketing. The radiated EM1 compliance limits for the U.S. are defined in Table 1.

Class A computing devices are used in a business, commercial, or industrial market and are not intended for use in the home. Class A devices need only be verified. Under verification, the manufacturer tests and labels the product. No filing is required.

Class B computing devices are marketed for use in the home and include personal computers, electronic games, and musical synthesizers. Class B devices must be certified.

Certification requires a detailed report of measurements, block diagram of the system, narrative description of the product's operation, user's manual, photographs of the product (inside and out), engineering drawings and schematics, and proposed equipment identification (FCC) label. The application and its required fee is shipped to the FCC and you wait for the Grant of Equipment Authorization.

Peripherals (i.e., printers and modems) fall under the same classification as the computing devices they will be connected to. Computer plug-in cards which have connections to an external device must also comply.

Exemptions-did someone say exemptions! Any computing device used exclusively in a motor vehicle is exempt. And, then there's NC machines, textile dryers, robotic systems, laboratory and automotive test equipment (hmm.. .the compliance test equipment is exempt!), appliances using computing devices like dishwashers, sewing machines, and power tools as well as specialized medical treatment equipment (e.g., CAT scanners) that are also exempt.

THE UMPIRES

To assure adherence to the rules, the FCC has defined test-site construction and test procedures, which include the periodic calibration of all test equipment. These regulations ensure that testing is fair to all independent of the compliance testing laboratory.

Under Part 15, Subpart J, these facilities test products for both power-line-conducted emissions and radiation. If all measurements are below the prescribed limits, documentation is submitted for FCC approval.

The costs for testing can be $1,000–2,000 per day, and they rarely exceed one day unless the emissions exceed posted limits. At which point, you either take the product back and work on it without the aid of test equipment or pay for more time and work on it there while the meter's running (the last option only works when the facilities don't have the next time slot filled).

It is rare that a doctor of EM1 is available, can read your charts, come up with a solution, apply it, retest, and get you on your merry way all in one

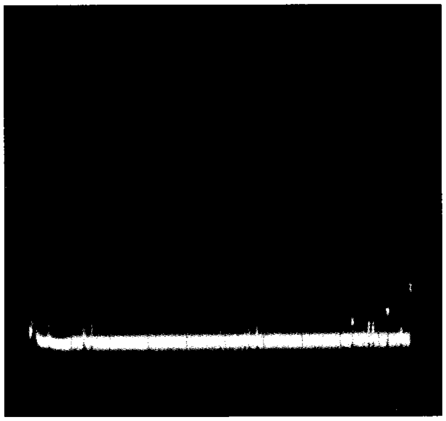


Photo *J--Background EMI sources are no cause for alarm.*

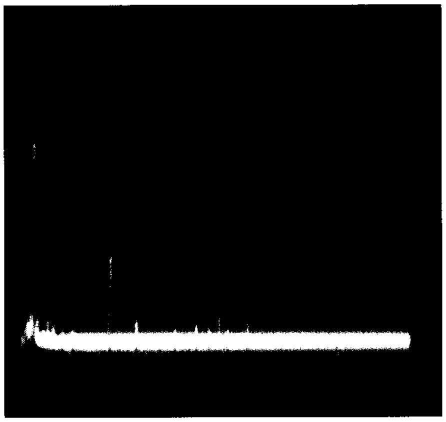


Photo *4-Here, the main processor's crystal is seen (fundamental frequency is 18.432 MHz).*

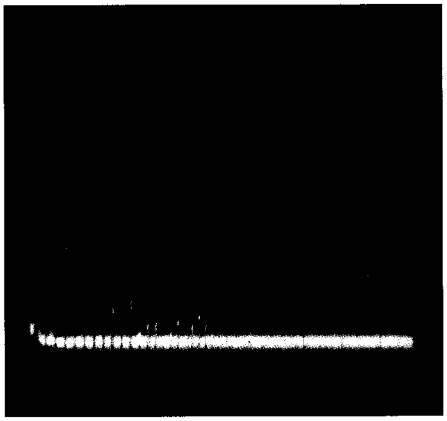
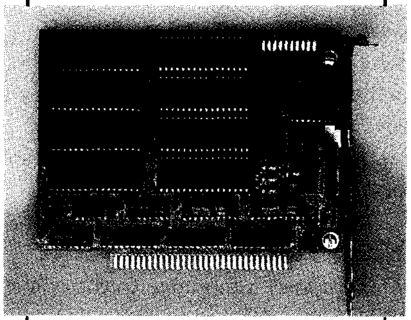
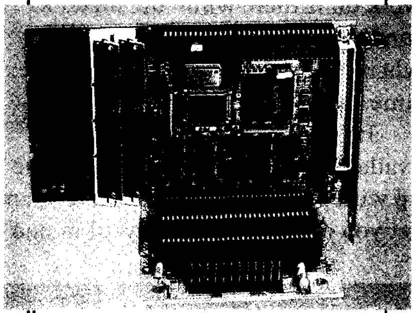


Photo **5—***This spectrum reveals normal bus emissions when probed about 1" from the PCB.*





day. Oh sure, they might tack in a capacitor or two, throw in some ferrite, or cover your product with aluminum foil. But, by the time you leave, you may not recognize your product, never mind reproduce it.

Anyway, it takes at least another day to retest the product once you've implemented the changes into the next production prototype.

REMOVING THE BLINDERS

Going into a compliance test cold asks for delays. Compliance must be on your mind in the earliest design meetings. Every item added to your design should be considered as a potential radiator. Track all the frequencies used in the system such as oscillators, tuned circuits, and processor control signals (e.g., ALE].

The majority of problems occur at other than the fundamental frequencies. In many systems, fast edges and wide bandwidths are necessary, but don't be fooled into thinking faster is better. Fast transition times spell trouble via PCB traces and I/O cables.

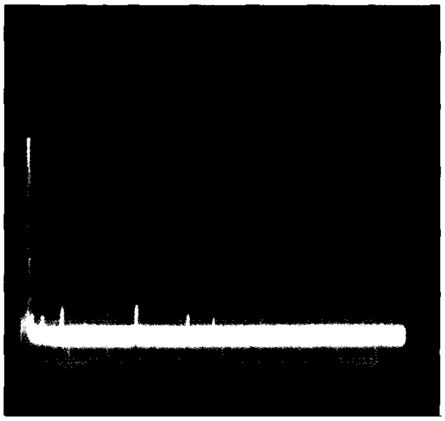


Photo 7—Shielding the inside of plastic enclosure brought Homerun's EMI under control.

Rise and fall times can be controlled by the logic family used. CMOS, on one hand, has rise and fall times on the order of 50 ns and a bandwidth of 6 MHz, whereas the LSTTL family is 10 times faster with a bandwidth of 60 MHz. Don't overdesign. Use slower parts when possible so you can nip harmonics before they become a problem.

Pay attention to the I/O lines exiting your product. The wiring used to attach peripherals (e.g., sensors, motors, printers, telco, etc.) must be in place during testing. These lines can become radiators from I/O signals or through conduction from other radiators. Use limited-bandwidth drivers when possible or you may need to filter each I/O. Always route wiring and cables away from noisy areas in the system to reduce the possibility of conduction.

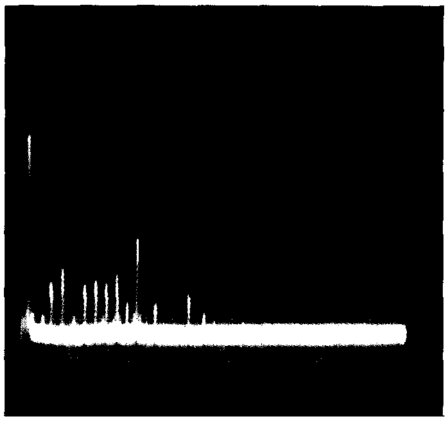


Photo 6—In 1985, the Homerun Home Control System system failed EMI compliance on the first try with just a plastic enclosure.

PCB designers should understand where potential hot nets are located and keep these interconnections as short and direct as possible. Crosstalk from noisy neighbors can be reduced by keeping parallel runs as far away as is physically possible.

Increased supply current flows though gates during output transitions. This extra current, drawn through the V_{CC} and V_{DD} power traces, can cause a voltage drop in the traces themselves. This drop raises V_{DD} (and lowers V_{CC}) at the gate, causing ground bounce. If these bounces exceed noise margins, a potential confusion in logic states can

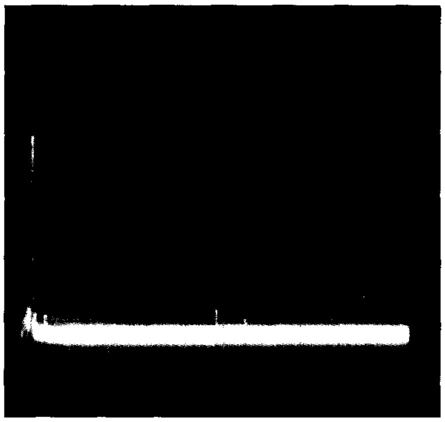


Photo 8—The programming supply on the SEP27 radiates switching hash when the required current changes during programming.

develop between connected gates in different packages. Decoupling capacitors are used to help supply the momentary additional current.

Multilayer PCBs offer the use of ground and power planes. These should be placed on the inner layers to provide superior high-frequency decoupling and shielding. The inner planes can be split for devices which require multiple power supplies, but care should be given to prevent overlapping planes when noise can be capacitively coupled.

Above all, the design team must be familiar with the radiation limits their system must comply with.

FIELD STRENGTH

Three sets of units are commonly used for EM1 measurement: decibels below milliwatt (–dBm), decibels above a microvolt (dBμV), and microvolts (μV). To convert between them, use the following formulas:

$$20\log_{10}(\mu V) = dB\mu V$$
$$-dBm = dB\mu V - 107$$

The test setup for radiated emissions usually consists of a rotating platform which allows the DUT (device under test) to be spun 360”. The receiving antenna is generally located 3 m from the DUT and can be raised up to 4 m above the ground plane built into the floor.

The procedure during each tested portion of the frequency band is to rotate the DUT and to move the receiving antenna to positions causing the greatest radiated energies to be received. A spectrum analyzer measures the emissions and displays them as the testing continues through the spectrum from 30 MHz to 1 GHz.

It is important to note that other sources of radiation such as local radio and television stations, airport control tower, cab companies, or other transmissions sources show up in the tests. These transmissions must be identified by the testing laboratories so they are not confused with the DUT. Peak signals originating from the DUT are recorded and converted into μV/m. If all peaks fall beneath the test limits, the product passes.

Conducted emissions are measured through the use of a line-impedance-stabilization network (LISN). This device is placed between the DUT and the AC power source, so there’s a known line impedance on which to base its measurements. The LISN provides a direct connection for the spectrum analyzer and may also provide filtering to assure the only emissions measured are coming from the DUT. The tested frequencies are between 450 kHz and 30 MHz. If all the measured peaks remain below the test limits, you go home happy.

If just one of those nasty little harmonics is above the maximum, “Buzzt! I’m sorry, you lose.” If you’re fortunate enough to actually be present during the testing, you may be able to wiggle, poke, and shield different areas of the system to try to identify where that little nasty is coming from. If you’ve brought your shop along with you, you might even have a chance to fabricate a cure. Otherwise, say good-bye to the nice people and schedule another appointment on the way out.

a)

Class A	Class B	Frequency (MHz)
300 μV/m	100 μV/m	30–88
500 μV/m	150 μV/m	88-216
700 μV/m	200 μV/m	216-I 000

b)

Class A	Class B	Frequency (MHz)
1000 μV/m	250 μV/m	0.45-I .6
3000 μV/m	250 μV/m	1.6-30

Table 1—*The* FCC *places* ***limits*** *on the emissions energies permitted for each frequency band.*

MONITOR YOUR PROGRESS

All radiation begins as either magnetic or electric in nature. The magnetic field or H-field is produced by high currents, whereas the electric field or E-field is created by high voltage. Radiation is often a combination of each.

The area in which the radiation can be individually detected is referred to as the *near field.* As the E- and H-fields travel through the air, they reach a point of equilibrium known as the *transitional region.* At 50 MHz, this region is about 1 m. Higher frequencies reach this region sooner than lower ones. The region after transition is referred to as the *far field.*

Compliance measurements, taken at a distance of 3 m, are past the transition region and into the far field for all frequencies of interest. However, the problem is that from 3 m you can hardly see the product, never mind its potential hot spots (radiation point).

What, you don’t have $10–$20k for a spectrum analyzer and no room for a formal test site?

This is where near field detection can help. Add these two items to your tool box. The first, shown in Photo 1, is the Spectrum Probe from Smith Design. It is an RF spectrum analyzer in a probe that you attach to your O-scope. The second, ScanEM-E and -H (see Photo 2), is a self-contained wideband emissions detector by Credence Technologies.

Using the Spectrum Probe is easy but requires an oscilloscope with at least a l-MHz bandwidth for display. The Model 107 probe puts out a logarithmic amplitude signal (vertical axis) on a triggered sweep displaying l-100 MHz (horizontal axis). (Model 255 displays the lower range of 30 kHz to 2.5 MHz.) Although not calibrated, it gives you a good look at what kind of emissions your system puts out.

The display’s dynamic range is 60 dB minimum with a logarithmic linearity of ±3 dB. The spectrum is flat to ±2 dB in Model 107’s 5-100 MHz range and 50 kHz to 2.5 MHz for Model 255. Included with the Spectrum probe is a small booklet filled with techniques on how to use the probe with a whole slew of circuits. This information helps you investigate signals from AC line conduction to ultrasonic and infrared. Optional RF current probe attachments make it possible to sniff out individual circuit traces.

The ScanEM probes, one for E-field and one for H-field, are truly all in one. A hold-to-test button conserves battery life while a single pot adjusts the threshold or background noise level. This level is displayed on a five-LED bar graph and through the varying pitch of the tone generator. The wideband emission detection has a resolution of about 1 dB per LED. The E-field probe has a bandwidth of 5 MHz to 1 GHz and the H-field probe has a bandwidth from 100 kHz to 100 MHz.

HANDS ON

Having gone through a half dozen certifications over the past 10 years, I have some real test report data. I am fortunate to also have the original products which produced emissions data. Let's see if any correlation can be made between the test results and what I can see using these new tools.

During a compliance test, the product is first analyzed in a completely shielded room. Since no external radiation is in the room, the technician gets a product's emissions signature without other sources adding to the confusion. This unnatural environment cannot, however, be used for the certification testing.

For informal testing, I'll use the opposite approach. I'll take a signature of the ambient conditions and then look only at the changes in signature when the DUT is turned on. I'll experiment on three products: the Homerun [a home-control system from 1985), the SEP27 (a serial EPROM programmer from 1987), and SpectraSense 2000 (a home-control system from 1994).

The Spectrum probe is connected to a Tektronix Model 2445A oscilloscope. A 10" piece of #18 wire is added to the probe's tip and the antenna/probe is suspended over the product at a height of 1 m. At this distance, signals less than 50 MHz would be considered far field and those greater than 50 MHz are near field.

Remember near field signals, especially those within inches of the product, are electric and magnetic in nature and will not correspond directly to what would be received in a far field measurement.

We are looking for relative signals here. The Spectrum probe should show the frequency ranges the energies are concentrated in. The ScanEM probes, on the other hand, should pinpoint individual circuit hot spots without additional equipment.

Photo 3 shows the background signals common to our locale. The spectrum includes AM, FM, and TV as well as the occasional taxi, aircraft, and police communications. The AM radio stations bounce up and down because the carriers are amplitude modulated. The police, taxi, and other spurious signals can give you headaches if you don't record their existence and they pop up during a test.

When I looked for emissions on the SpectraSense 2000, which is a single-board HCS that provides the equivalent funtionality of a Supervisory Controller, PL-Link, DTMF, and two BUF-Term boards (plus a little more], I could see little difference from the common background signals. SpectraSense's NEMA 2 enclosure shields against the three separate crystal clocks used within.

Even with the enclosure open and the Spectrum probe held within an inch of the crystal, the emissions are low thanks to the multilayered PCB's ground plane. Photo 4 shows the radiation spectrum from a clock source (18.432 MHz). Photo 5 indicates a fairly normal emission from normal bus activity, again with the probe close to the PCB.

One thing a home control system is usually responsible for is X- 10 transmissions. The TW523 power-line interface induces a 120-kHz carrier on the AC line. Transmissions of this carrier can be clearly seen using the Model 255's current probe on the TW523's AC line output.

Now let's step back a few years to Homerun. The first prototype was housed in an unshielded, slanted, plastic enclosure with a full ASCII keyboard. Photo 6 shows the emissions at 1 m with a 10" antenna. All probes (Spectrum and ScanEM) showed maximum signal strength emanating from the video section of Homerun when a close range scan was executed over the entire PCB. This system failed emissions tests.

A second prototype was certified once it had been shielded with a spray-on coating of nickel acrylic. The shield and keyboard frame was connected to system ground and Photo 7 shows a dramatic decrease in emissions. Since this product used only a double-sided PCB, it had no internal ground plane.

Close-range scans are most effective for identifying hot spots. The SEP27 has two such hot spots. The normal clock oscillator running at 11.0952 MHz and the switching power supply used to create +30 V from +12 V. Photo 8 shows a large amount of hash associated with the switcher's inductor. Metal endplates and a conductive coating on the box shielded this product nicely.

A CLEANSING BREATH

I dislike the unknowns associated with a product that must undergo FCC-compliance testing. Sure, having gone through it makes it easier, but I still feel visually handicapped.

Until now. These tools give precious insight into a previously dark world. You have a household full of products that have passed the test. Go look at the portable phone, the kids' video game system, or your PC. You'll get a good feel for what's hot and what's not!

Remove those blinders and enter compliance head-on. □

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.

SOURCES

Smith Design
207 E. Prospect Ave.
North Wales, PA 19454
(215) 661-9107
Spectrum Probe

Model 255	$279
Model 107	$249

Credence Technologies
350 Coral St., Ste. 2
Santa Cruz, CA 95060
(408) 459-7488
Fax: (408) 427-3513

ScanEM-E	$140
ScanEM-H	$145

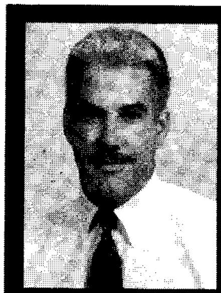
IRS

419 Very Useful
420 Moderately Useful
421 Not Useful

SILICON UPDATE

Tom Cantrell

PLDCONfusion



A PAL is a PAL, right? Not these days. Programmable logic has taken off, and with it comes many of the same dilemmas faced when trying to select a microprocessor. Tom tries to make some sense of the confusion.

I often regret that I don't get to do a lot of hands-on engineering these days, Instead of pushing the state of the art, I seem to end up pushing paper.

Engineers often lament their lot in life, complaining of a lack of respect and longing for the big bucks lawyers and doctors make. Proposed solutions range from ABA- or AMA-like licensing to calling on Hollywood to make a TV show about engineers ("this week Ned's soldering iron explodes and Edna's PC contracts a virus after a wild night on the Internet").

Sure, engineers don't make as much as doctors. Just remember that the average artist, musician, lifeguard, ski instructor, or even West Coast Editor doesn't make as much as the guy that hauls the trash. You don't have to be Milton Friedman to contemplate the idea that fun-and-interesting jobs require less monetary motivation.

A common argument that testing and licensing would enable engineers to achieve the status of lawyers recalls the adage "be careful what you wish for 'cause it might come true." It doesn't take more than a few minutes of watching the O.J. circus to see what I mean. Yeah, you'd have to pay me $300 an hour to sit there day after day with a straight face too.

Nevertheless, there are times when even an engineering exercise gets messy, fraught with confusion, political partisanship, and sound-bite marketing.

I recently attended the ***EE Times*** sponsored PLDCon in Santa Clara to scope out the PLD market thereby saving you, our illustrious readers, the hassle. Confronted with a sea of strange acronyms and architectures, not to mention head-spinning tool conglomerations, I came away with little understanding (but a truly back-breaking stack of literature)

So, rather than attempt a pithy summary, I'm just going to dump the data and leave the engineering exercise of figuring out what the heck it all means to you.

RISC VERSUS CISC

One way of trying to make sense of it all is to map the twists and turns along the PLD roadmap in light of the somewhat more mature microprocessor marketplace. As you'll see, there are both striking similarities and weird differences.

As for micros, which evolved from a pair of primal contenders (the Intel 8080 and Motorola 6800) in the '70s, the PLD market was pretty much summed up in the '80s by the PAL (originally developed by Monolithic Memories, which was later acquired by AMD) and the Xilinx SRAM-based LCA (Logic Cell Array).

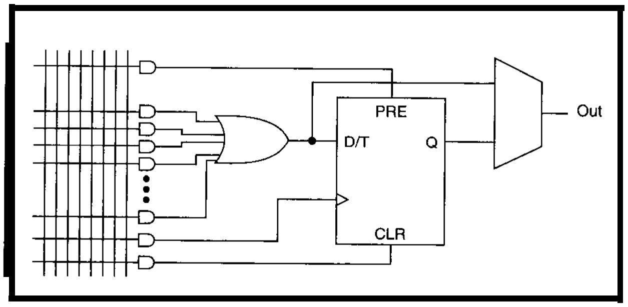


Figure 1-A generic *CPLD logic cell reflects its PAL sum-of-products heritage with an array of AND gates feeding an OR gate. With a large number of possible inputs (20 to 80 is typical), a PLD is especial/y wel/-suited to hand/e wide functions (such as decoders and counters) within a sing/e level of logic for speed.*

Just as micro architectures proliferated in the '80s with everybody hopping on the bandwagon, the PLD market of the '90s is characterized by an explosion of parts, concepts, and hype. The conference directory lists dozens of companies making chips with dozens more offering design tools.

Roughly (very roughly) speaking, architectures tend to be classified as coarse or fine grained, a heritage traceable to the original PAL versus Xilinx schism.

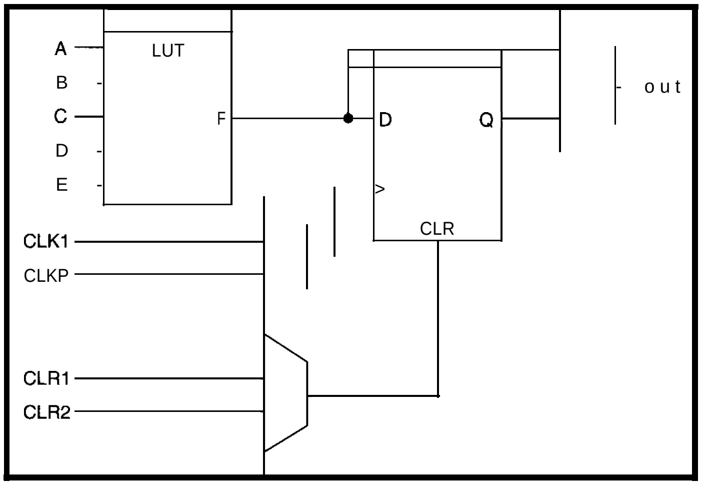


Figure 2—*By* contrast, an *FPGA cell* is *much* simpler (but there *are more of them),* featuring a few *combinatorial inputs typically fed as* addresses *to a RAM look-up* fable. The *data stored at each address defines the cell's logical function. With* more cells *and registers per chip,* FPGAs *are* well-suited *for* synthesis *and pipelinable applications.*

The paper "When CPLDs Might be the Best Solution to Your Design Problem" by Chuck Tralka of ICT Inc. attempts to quantify the difference by highlighting the variances between a coarse-grained CPLD (Complex Programmable Logic Device) and the finer-grained FPGA (Field Programmable Gate Array).

The best way to visualize a CPLD is as a small collection of PALs combined with a simple centralized routing matrix. Indeed, the generic sum-of-products CPLD logic cell (see Figure 1) is little changed from that of the original PAL.

A fine-grained FPGA like Xilinx consists of a larger number of simpler cells connected by a sophisticated hierarchical routing scheme. As shown in Figure 2, a generic FPGA cell is often implemented as a small look-up table, which can take advantage of finely tuned memory-manufacturing techniques.

The difference is aptly summarized in Table 1, which compares the characteristics of a sampling of 84-pin CPLDs and FPGAs and yields some insight into which should be used when. Though CPLDs have fewer cells, each is able to do more with six times the fan-in and three times the combinatorial capability (measured as SOP, i.e., sum of products). Thus, CPLDs are well-suited to handling wide [high fan-in) functions such as address decoders, counters, and comparators.

Compared to CPLDs, FPGAs are register rich by necessity to let complicated functions be scattered across multiple cells. Indeed, if your application is amenable to pipelining (i.e., throughput is more important than latency), FPGAs may offer the best fit. Of course, extra registers are a plus when it's time to whip up some RAM or a FIFO.

Much of the micro's RISC versus CISC dogma is echoed in the CPLD versus FPGA wars. Notably, when it comes to synthesis, FPGA proponents argue (as do RISCers) that it is easier for a compiler to target a simple, regular cell (instruction) than a more baroque selection. CPLD proponents echo the CISCers' code-efficiency arguments, pointing out that silicon (code space) reduction is possible with more specialized cells and dedicated wiring.

Device	Number of Cells	Fan-in/Cell	SOP/Cell	Number of Registers	Number of I/O Pins
EPM7064	64	36	32 (max)	96	68
MACH130	64	26	12 (max)	64	70
ispLSIlO32	128	36	20 (max)	192	72
Average CPLD	85	33	21	117	70
Median CPLD	64	36	20	96	70
ACT1010B	295	8	4	147	57
XC3020	64	5	16	256	64
AT6002	1024	3	2	1024	64
Average FPGA	461	5	7	476	62
Median FPGA	295	5	4	256	64
Ratio of Average CPLD/FPGA	0.18	6.60	3.00	0.25	1.13

Table I--The *difference between CPLDs and FPGAs is quantified by this architectural comparison of 84-pin chips.*

TOOL TUSSLE

Speaking of synthesis, there's no doubt that HDLs (Hardware Description Languages) such as VHDL and Verilog are poised to make a move from the ASIC (i.e., gate array) to PLD arena. As PLDs expand from thousands to tens of thousands and ultimately hundreds of thousands of gates, schematic entry is starting to run out of gas.

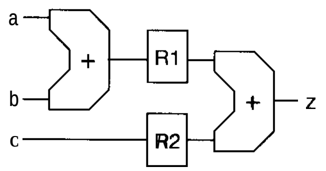
For those not familiar with the concept, synthesis purports to translate an ASCII behavioral model into the corresponding low-level net list (gates and connections). The beauty of so-called top-down design techniques is that the behavioral model can be completely simulated (or even emulated) to debug and test the design long before the chip sees the light of day.

While synthesis is becoming routine for 100k+ gate ASICs, questions still abound about when synthesis for PLDs will shift from the bleeding edge to the more manageable leading edge. Move too soon, and you spend a lot of bucks on tools of questionable fortitude and end up with a funky, slow chip. Procrastinate too long, and you can flail in a sea of schematics while your competitors sail smoothly by.

Figure 3, extracted from Warren Savage's paper "Making the Leap to HDL/Synthesis," shows an example of a three-input adder written in Verilog. A key issue for synthesis is how and/or whether to insulate the behavioral model from low-level implementation details. For instance, the model in the figure does not indicate whether a simple ripple or a faster pipelined look-ahead adder is generated. Most tools try to make a good guess based on a user's specified size, speed, or timing constraints.

While there's no doubt HDLs ultimately prevail, there's certainly dissent over which-HDL, Verilog, or VHDL-will dominate. When I first covered synthesis in 1990 ("VHDL—The End Of Hardware?" INK 17), I blithely predicted VHDL would win. The bad news is I was wrong since Verilog, exploiting its early lead, remains a contender. The good news is that most major tool suppliers courageously offer both VHDL and Verilog. The bad and good news is that you must figure out which way the wind is blowing and make a decision.

The mutual interest in synthesis by both the ASIC and PLD crowd reflects another common need.



```
//pipelined 8-bit adder circuit
// * adds 3 operands: a,b,c
// * pipeline needed for speed

reg [7:0] R1,R2;
always @ (posedge Clk)
   R1 = a + b:
always @ (posedge Clk)
   R2 = c:
wire z = R1 + R2;
```

Figure 3—*Much* as a *C program is compiled info machine code, logic synthesis converts an HDL (in* fhis *case Verilog) program info the equivalent nef list.*

Customers would like to easily migrate a single behavioral model from PLDs for early production runs to lower-cost ASICs once the design is proven and in volume production. Indeed, they'd like to easily migrate between any variety of ASIC and PLD architectures and suppliers at will in search of the best deal. Just plug in a new disk, punch a few keys, and voilà—not!

Similar to the widely touted concept of C program portability across processors, there's a fair amount of hope (not to mention hype] involved in migration among and between PLD and ASIC architectures.

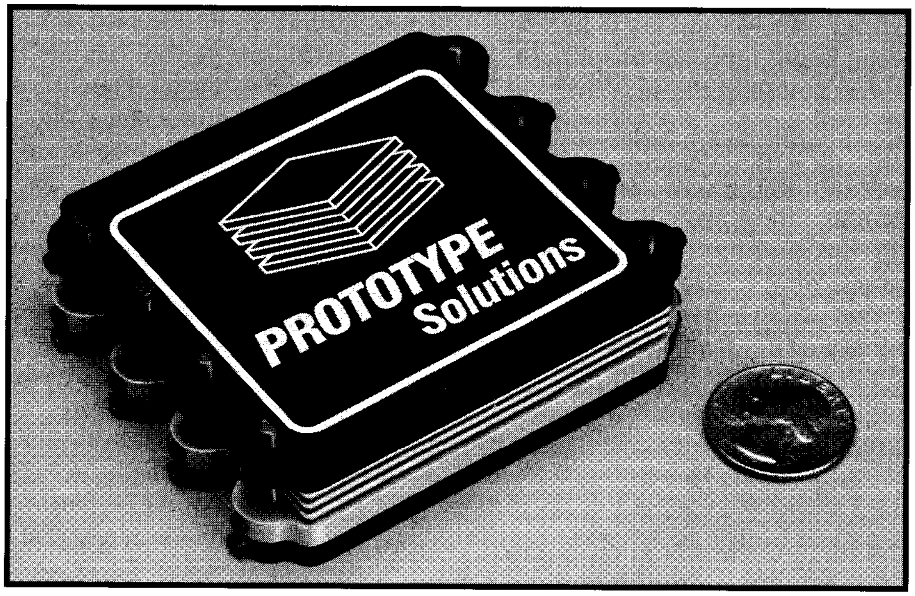


Photo I--The *Prototype* Solutions *STANDin exploits 3D packaging fo boost density info the 100k gates, 1k pins stratosphere.*

In the paper "HDL Methodology Offers Fast Design Cycle and Vendor Independence," Joseph Cerra of Wellfleet Communications proposes a solution based on a BIOS-like separation of core and wrapper modules with the latter exploiting or isolating the low-level vagaries of a specific device. Migration hassle is minimized without incurring the transistor bloat of transparent portability.

It is feasible to migrate between PLDs and ASICs via synthesis as long as you are prepared to take the speed and size hit the generality of a PLD incurs. For instance, the Verilog adder described earlier in Figure 3 consumes almost twice the gates and runs nearly 10 times slower when retargeted from gate array to FPGA.

FIELD-PROGRAMMABLE EVERYTHING

The subject of in-system programmability, covered in "Update Your System On The Fly" *[INK* 44), has taken hold with a vengeance. As I discussed, the original concept of field programmability typically meant the part could be blown (like the original fuse-based PALs) with a low-cost programmer.

By contrast, in-system programmability, as the name implies, means a device can be configured without removing it from the system, which is great for bug fixes. Clever designers are now recognizing the benefit of dynamic reconfigurability so that the chip can be repeatedly and arbitrarily reprogrammed on a whim.

The evolution is easily seen by comparing an Aptix part mentioned in *INK 44* with a new competitor from I-Cube. Interestingly, both of these are emerging variants of PLDs, best described as PWDs (Programmable Wiring Devices).

Both the Aptix and I-Cube parts offer a zillion pins that you connect by programming SRAM bits. However, the Aptix part (like an FPGA) needs to go through a lengthy routing exercise to come up with the corresponding bitmap. It might be possible to go in and twiddle a few connections after the fact, but it isn't straightforward or really the intention of the part.

By contrast, the I-Cube chip is designed from the ground up for dynamic operation. Configured as a true nonblocking crossbar switch, connections can be made or broken at will and at high speed (in as little as 20 ns). Direct, rather than routed, connections offer throughput of up to 133 MHz (i.e., 6 ns in to out) and the delay for each signal is identical.

Thus, while the I-Cube part can also serve PWD applications, it's better suited for real-time programmable switching of grouped signals as might be found in any variety of high-speed data blasters (i.e., LAN, digital A/V, multiprocessor, etc.). Compared to PWD, these applications may not need to connect a full circuit board's worth of signals. So, I-Cube offers parts with as few as 32 pins and refreshing single-digit price tags.

Combine the programmable gates of a PLD with the programmable wires of a PWD, and you've got a complete programmable system. That's exactly what Prototype Solutions offers with their "STANDin" parts (see Photo 1).

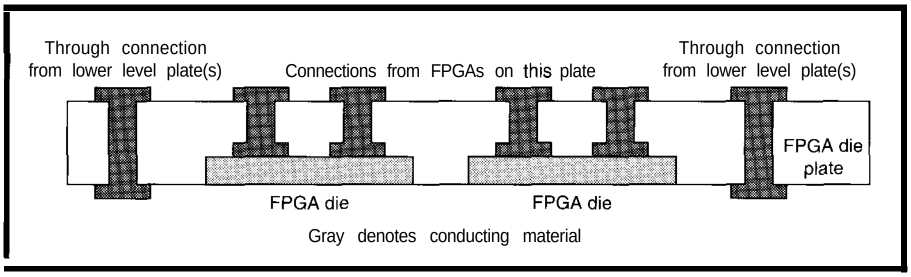
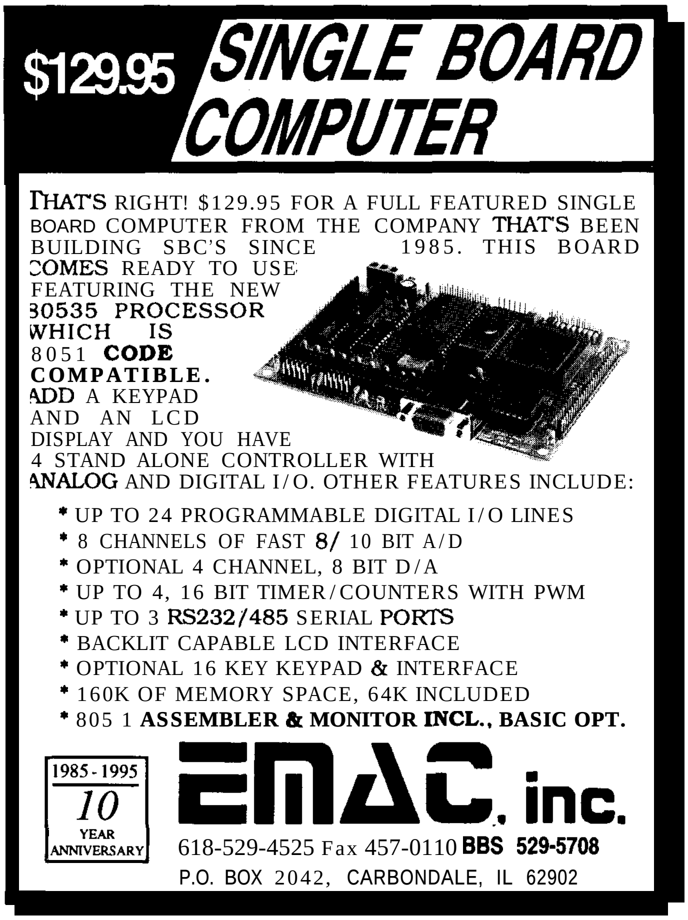

Through connection from lower level plate(s)
Connections from FPGAs on this plate
Through connection from lower level plate(s)
FPGA die plate
FPGA die
FPGA die
Gray denotes conducting material

Figure 4—*The STANDin is constructed as a series of stacked plates each containing multiple FPGA dice.*

As shown in Figure 4, the unit consists of stacked plates, each containing multiple Xilinx FPGAs, that are 3D interconnected with gold conductors. Larger units feature up to 24 FPGAs (240k gates) and as many as 1,280 pins, a rather formidable amount of punch in a small form factor.

To establish the interconnect pattern, a top plate, consisting of an XY wire grid, is programmed by selective cutting of the wires with a laser. The wiring is quite versatile, providing nearly total freedom in interconnecting the FPGAs and package pins. Since the PWD is real wire in this case, interconnect delays (a measly 1 ns or so) aren't a problem.

The technology is well-suited for prototyping a chip and shares a common design concept with the full-fledged ASIC emulators from companies like Quickturn. Prototype Solutions points out that Quickturn users can use that system for development and then make a few prototypes to pass around. The otherwise daunting $10k price for a STANDin doesn't sound so bad considering the $100k+ for a full-fledged ASIC emulator. Should a significant design change occur (remember that minor changes are likely handled by FPGA reprogramming), a new top plate can be zapped for only $500. The company also hopes volume production will enable the



STANDin price to fall from dimes to pennies per gate.

Another emerging application well-suited to multichip FPGA arrays are so-called reconfigurable computing elements, which can supplement, if not replace, a micro or DSP. Why bother with a C program and processor if a similar HDL program can be compiled directly to hardware? The conference proceedings contain a number of papers addressing never-enough-MIPS applications like 3D graphics processing and image recognition.

Watch out though. While an FPGA has no problem with simple adders and counters, more complex math functions can quickly bring it to its knees. Rod Dewell's "Numeric Implementations using Programmable Logic" highlights the problem by showing how a Xilinx XC4010 (10k gate) FPGA can easily handle up to 50 16-bit adders, but barely a single 16-bit multiplier. Furthermore, the multiplier only runs at about 6 MHz (170 ns), not a heck of a lot faster than the latest DSPs or RISCs.

The paper makes the reasonable proposition that the best result is likely obtained by the judicious combination of general-purpose FPGAs and special-purpose numeric circuits. Infinite offers the RAD5A4 (depicted in Figure 5), which is kind of a coprocessor for FPGAs that contains four ALUs and can ADD, MLT, and MAC on the order of 10 times faster than an FPGA-only implementation. With each ALU operating concurrently, a [1x4] x [4x4] matrix multiply blows by at a speedy 25 MHz.

SO MANY CHIPS, SO LITTLE TIME

There's nothing I'd like more than to spend a few weeks (or months or years) wading through stacks of literature. As it is, I'll only be able to give you a brief sampling of the cornucopia of PLDs I encountered.

AT&T offers the 3000 series of Xilinx-compatible parts. There's some concern about the fact that their former development tool supplier was just acquired by-you guessed it—Xilinx. Meanwhile, their new ORCA (Optimized Reconfigurable Cell Array) family is also SRAM look-up-table based, but the SRAM is easily configured for memory as well. ORCAs also feature nybble (rather than bit-oriented) routing and copious I/O (up to 480 pins) for bus-oriented applications.

Actel continues to offer a broad line (ACT 1, 2, and 3 series covering from 1k to 10k gates) of fine-grained FPGAs. Their pioneering antifuse interconnect scheme (as the name implies, it represents a connection which is grown rather than blown), though only one-time-programmable, is both fast and very routable to maximize gate use.

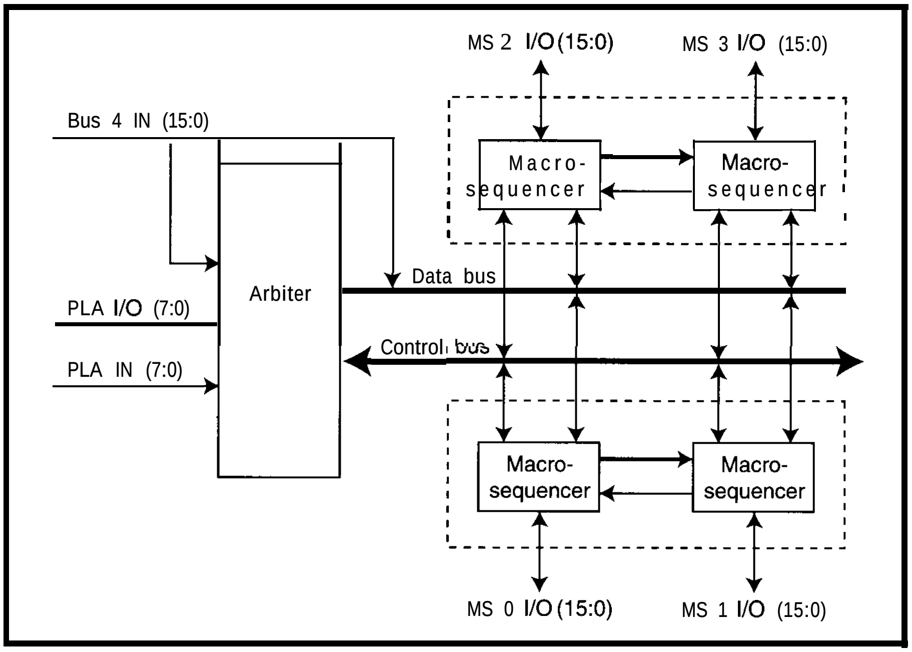


Figure 5—*The Infinite Technology RAD5A4 contains four 16-bit ALUs to offload gate- and speed-intensive functions from a connected FPGA.*

A new fine-grained entry comes from the GateField division of Zycad. What seems an unlikely source may make sense because Zycad, a maker of ASIC simulation accelerators, works with customers who constantly wrestle with routing and timing problems. The company claims high density (100k gates) and routability thanks to a flash-based switching element. Its architecture is also purported to be exceptionally synthesis friendly.

The Atmel AT6000 family takes dynamic in-system reprogramming to heart. Their cache-logic scheme is not only SRAM based, but it lets different functions be shuffled in and out of the chip in real time without disrupting register states, clocking, or I/O. If your application can be broken down temporally (i.e., functions operate independently rather than in parallel), the Atmel chip can mimic a fixed-function chip of correspondingly higher density.

Crosspoint Solutions was founded in 1989 on the intriguing idea of a true gate-array-like FPGA (i.e., they feature gate-level interconnect). The beauty of such a scheme is that it works with proven gate-array design tools and brute forces a solution to the PLD-to-gate-array migration problem. Unfortunately, they had a rather severe glitch: an antifuse that didn't ante up! Usually, startups that crash and burn are sent to that great fab in the sky with little left to show other than a tax writeoff. Crosspoint, to its credit, stuck with it and claims the problem is now fixed.

ICT continues to focus on their niche of low complexity, pin count, and budget for their EEPROM-based PLDs. With an ICT-supplied smart translator, their chips are notable for being able to mimic dozens of standard (i.e., JEDEC programming format) PALs, GALs, and EPLDs.

Xilinx, who invented the FPGA more than 10 years ago, continues to march forward, matching competitors blow for blow at the high end. Frankly, more interesting to me is their recent effort at the low end with chips like the 2k-gate, 84-pin XC5202. Xilinx chips, while nifty, have always commanded a high price. So it's pretty big news that the XC5202 is the first Xilinx chip to break the single-digit price barrier ($9 in volume).

The fun part, figuring out which chips and tools are best, I leave to you. It's surely an engineering exercise that will stretch your intellectual muscles to the limit. So, get on the horn, start dialing-for-DIPS, and don't give up 'til it all makes sense. •)

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.

IRS

422 Very Useful
423 Moderately Useful
424 Not Useful

CONTACTS

AT&T Microelectronics
555 Union Blvd., Ste. 21Q-133BA
Allentown, PA 18103
(800) 372-2447
Fax: (610) 712-2447

Actel Corp.
955 E. Arques Ave.
Sunnyvale, CA 94086
(408) 739-1010
Fax: (408) 739-1540

Atmel Corp.
2125 O'Nel Dr.
San Jose, CA 95131
(408) 441-0311
Fax: (408) 436-4200

Crosspoint Solutions, Inc.
5000 Old Ironsides Dr.
Santa Clara, CA 95054
(408) 988-1584
Fax: (408) 980-9594

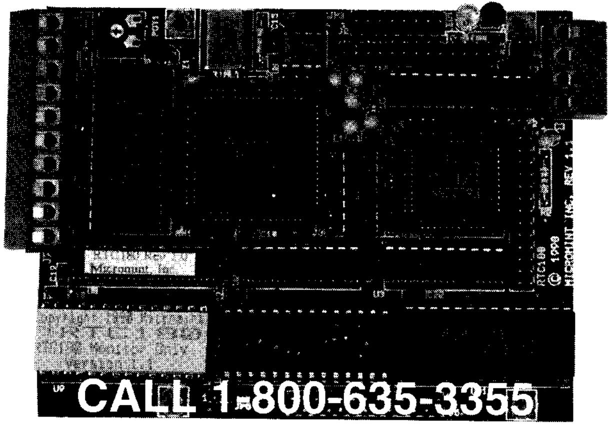
GateField
47100 Bayside Pkwy.
Fremont, CA 94538-9942
(510) 2495757
Fax: (510) 623-4484

ICT, Inc.
2 123 Ringwood Ave.
San Jose, CA 95 13 1
(408) 434-0678
Fax: (408) 432-08 15

I-Cube, Inc.
2328-C Walsh Ave.
Santa Clara, CA 95051
(408) 986-1077
Fax: (408) 986-1629

Prototype Solutions
19545 Von Neumann Dr., Ste. 135
Beaverton, OR 97006
(214) 690-4388
Fax: (214) 690-4388

Xilinx, Inc.
2100 Logic Dr.
San Jose, CA 95124
(408) 559-7778
Fax: (408) 559-7114

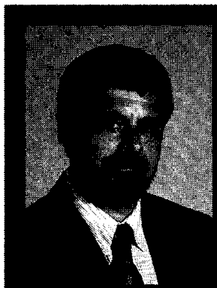


Power Management with the DS87C530

EMBEDDED TECHNIQUES

John Dybowski

Part 1: The Hardware



Instead of regulating power by turning the micro on and off, the DS87C530 can drop into a virtual zero-power state by shunting its minimal activities to internal oscillators. Even its real-time clock can go into an almost no-power mode.

Engineers are working full throttle at cutting system power requirements. Battery-operated systems must run for the longest possible time, often using the smallest available battery. And, the lust for low-power operation extends into nonportable, stationary equipment as well.

Designing truly low-power systems involves many disciplines. The subject is broad and exists in degrees—how low is low? A level of power consumption perceived as blissfully insignificant in one situation may be deemed totally unsatisfactory in another.

Systems running from battery power for long periods obviously require more engineering than those that merely operate at "reasonable" power levels. In the first setting, you need a comprehensive power-management strategy while in the other you can get away with simply populating a conventional design with predominantly CMOS parts.

It boils down to functionality versus economy. Given the contradictory requirements of the two strategies, it seems illogical to satisfy both with the same piece of equipment.

Until recently, this may have been true.

POWER-MANAGEMENT STRATEGIES

Traditionally, power management controlled consumption by selectively switching power to various system circuits such as communications drivers, data converters, and various other ancillary peripherals. Such a scheme keeps power consumption under control by removing power to various subsystems when services are not needed. However, this method of power management increases design complexity and component count.

Looking at the problem from another angle reveals other ways of addressing the same issues. This strategy is based on the principle of controlling consumption at the load instead of switching power at the source. This approach is more subtle in implementation, placing more emphasis on component selection, subsystem integration, and economy of execution.

CONTROLLING CONSUMPTION

CMOS devices draw extremely low quiescent current, consuming only significant levels when switching. Thus, overall current consumption is directly related to the processor's operating frequency. High-integration components minimize bus loading and the number of signals that must be taken off-chip. On-chip switching consumes considerably less current than that used by the heavy buffers driving a signal down a PCB trace.

The ultimate power savings is attained when the processor's oscillator is taken down to DC. In a carefully designed system, using this method could represent nothing more than leakage current in the microamps.

Of course, I'm assuming the system is designed properly. Should a chip select jam or a critical control line get set to the wrong state, current consumption could go up! There's more to it than just killing the clock.

A REPRESSIBLE CONTROLLER

A general-purpose single-board computer capable of operating at various power levels requires a highly integrated microcontroller. To be useful for development, the system should be usable in external memory mode. An economical implementation strongly favors an 8-bit implementation. A standard microcontroller architecture relieves the user from obtaining and mastering new software development tools. For high-level programming languages, the system should offer high-level performance.

Quite a number of modern microcontrollers possess the integration necessary for a minimal component count. However, a comprehensive set of power-management capabilities drops the number of candidates down to next to nothing.

Dallas Semiconductor's DS87C530 high-speed controller meets these goals with no appreciable increase in circuit complexity or cost. Although the DS87C520 is built around the same core and has the same basic features, the DS87C530 offers additional capabilities useful in certain types of data collection applications. With the exception of the built-in real-time clock and battery-backed RAM, all aspects of the subsequent discussion are relevant to the DS87C520 as well. Since the system gains its flexibility through the DS87C530's power-management modes, I'll begin with a brief overview of what they are and what they offer.

• Idle Mode

Idle mode on the DS87C530 is identical to that of the 8051. It halts processor operation but leaves the internal clocks, serial ports, and timers running. Normal operation resumes when an interrupt occurs. Although included for compatibility, Idle mode is made obsolete by the DS87C530's more advanced power-management capabilities.

• Stop Mode

Entering Stop mode puts the DS87C530 in the lowest power state possible. The crystal oscillator is stopped, which stops all internal switching, causing processing to cease. The DS87C530 exits Stop mode on reset or in response to an external interrupt.

Asserting reset terminates Stop mode, reinitializes the DS87C530, and causes program execution to commence from address 0. A more useful exit results from the assertion of an enabled interrupt. On interrupt, the DS87C530 simply resumes execution

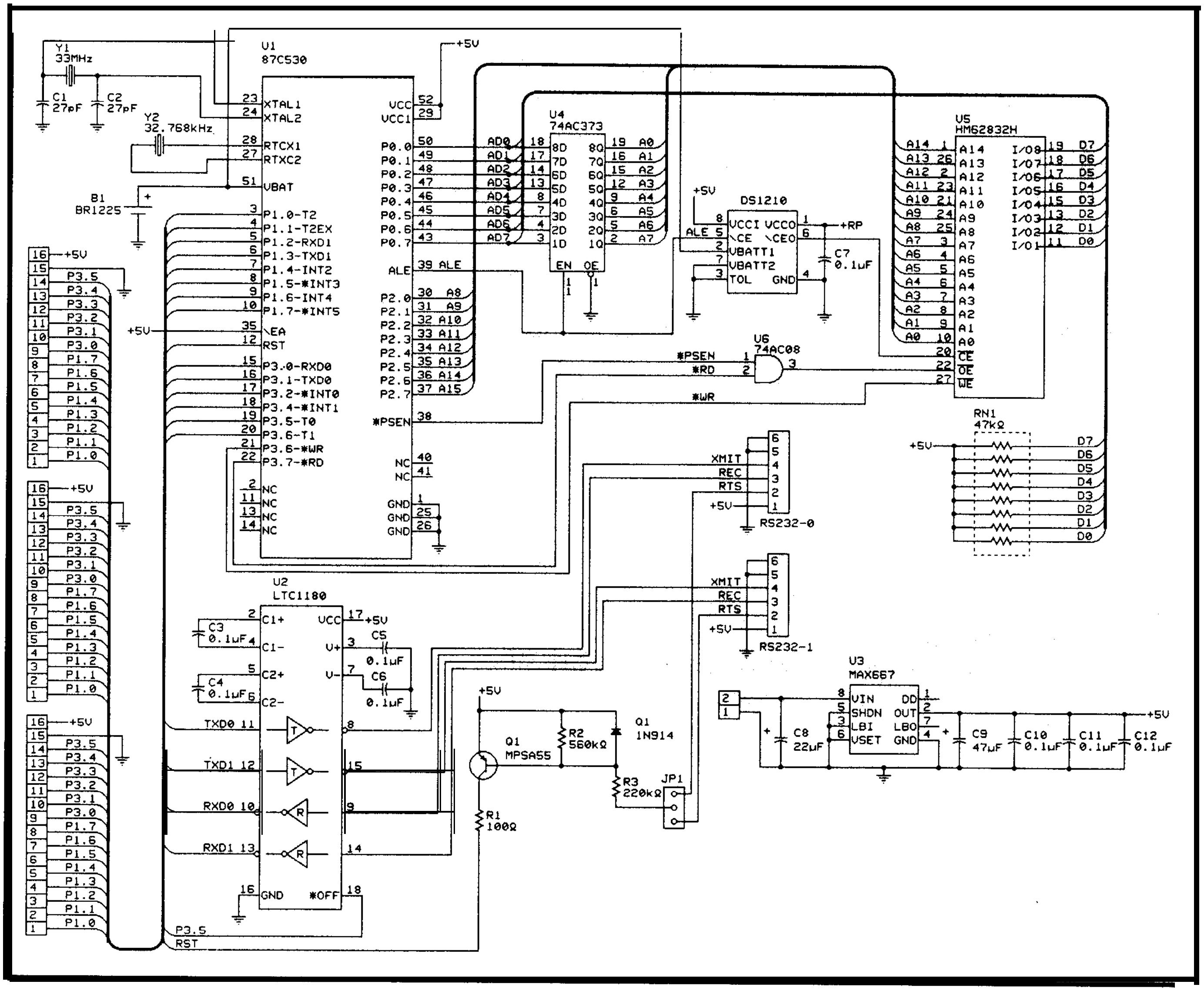


Figure 1-A general-purpose system based on the DS87C530 features an B-MIPS processing core and full power-management capability

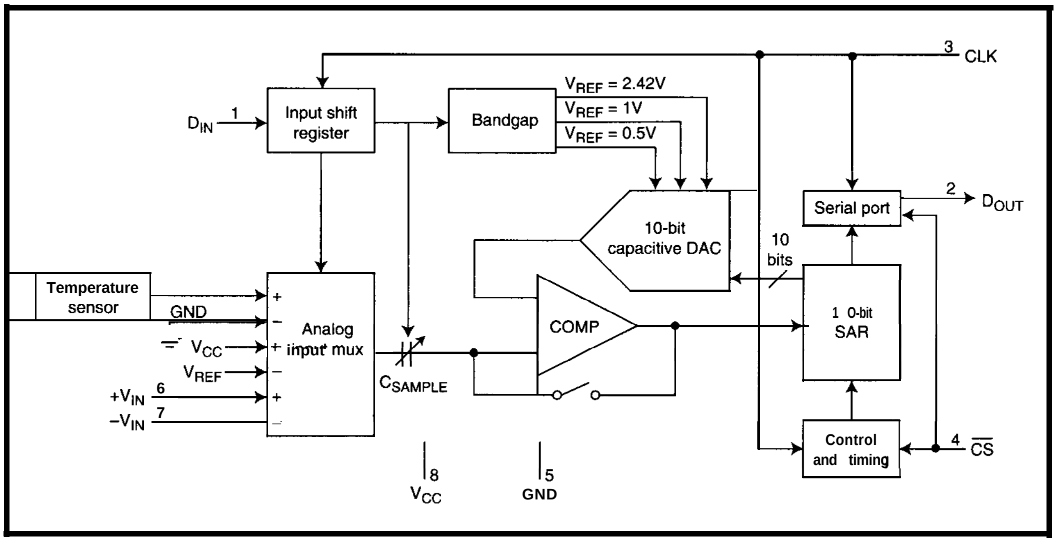


Figure 2—The LTC1392, which contains a temperature sensor, rail-to-rail differential input, and V_{cc} monitor, is a complete data acquisition front-end in an B-pin chip.

at the appropriate interrupt vector. Things weren't always this easy.

The original 805 1 required a full hardware reset to restart the processor. Some derivatives improved on this by allowing exit from Stop mode in response to an external interrupt. Usually, this method required programming the interrupt to be level triggered and dictated the external circuitry hold the interrupt until the crystal oscillator had time to stabilize. Processing effectively remained suspended, which was not an unreasonable restriction, since executing with an unstable clock can result in errant program execution and a loss of software control. Several milliseconds or more can elapse before the oscillator becomes stable.

Many applications that use Stop mode operate in burst mode. A typical data collection system might enter Stop mode while waiting for an external event to occur. With stimulus, the system resumes operation, takes some readings, and stores the results to memory. A return to Stop occurs in preparation for the next event. Unfortunately, with a conventional implementation, waiting for the oscillator can take much more time than processing the event.

To counter this potential waste of time, the DS87C530 incorporates a 4-MHz ring oscillator for fast recovery from Stop mode. Essentially a gate loop which relies on the circuit's propagation delay, the ring oscillator does not exhibit the stability of a crystal oscillator but can start up instantaneously, providing immediate resumption of processing.

Processing that is not timing dependent can be performed while running from the ring oscillator. When a stable time base is required, the software can take preparatory steps, wait for the crystal to come on-line, and then proceed with time-critical operations. To handle synchronization, status bits indicate the current clock source.

Another improvement (or correction, if you will) to Stop mode is worth mentioning. Like the 805 1, ports 0 and 2 continue to emit the information present when Stop mode is invoked. If the program is executing from external memory, you end up with a stuck chip select in your program memory chip unless you provide external gating.

The DS87C530 **pulls** \PSEN and ALE high (the 805 1 defined these pins as low). ALE is then used as a qualifying signal to ensure that no external memory devices are enabled while the system is in Stop mode.

. Power-Management Mode

This mode can also be thought of as clock-management mode. The correlation is natural since power consumption of CMOS circuits has a direct relationship to operating frequency. Previously, a lot of circuitry, not to mention engineering effort, was expended creating an effective clock-management structure.

Happily, power management along with a number of supporting features comes neatly integrated in DS87C530 silicon. Two levels of power management can be used in combination with either the crystal or ring oscillator yielding several levels of power control.

Power-management mode 1 (PMM1) forces the processor from its default 4-clock machine cycle to a 64-clock machine cycle. The crystal oscillator still operates at full speed, but all peripherals and instructions operate at a reduced rate. Resumption of full-speed operation can be invoked under software control or by using a special autoswitchback feature.

Power-management mode 2 (PMM2) operates under the same principle but offers greater power savings by dropping to a 1024-clock machine cycle. All other aspects of PMM2 are identical to PMM1.

Power-management modes are best used during periods of reduced throughput. Consider a system that operates in burst mode. Although PMM1 consumes 4 times less current than at full speed, processing takes 16 times longer. Consequently, short-burst operations should be performed at full speed for maximum efficiency. The same relationship holds true whether running off the crystal or ring oscillator.

• Full-Speed Switchback

In PMM, the DS87C530 can be set up to use an automatic switchback to quickly restore high-speed operation. This feature is event driven and requires no software intervention. The following sources can be enabled to trigger a switchback:

- external interrupts
- serial start-bit detection

- transmit buffer load
- reset (watchdog timer, power-on, external)

In general, switchback occurs in response to an interrupt. Note, however, that this introduces problems for serial communications. For the serial port to operate properly, the DS87C530 must be operating at full speed. If switchback is coupled to a serial interrupt, the transmitted or received data is hopelessly corrupted by the time the interrupt occurs.

The receive problem is overcome by initiating switchback on the falling edge on the RXD pin. Full-speed operation resumes on the next machine cycle in time to capture the start bit and successfully receive the data byte. Instead of being dependent on the serial interrupt being enabled, the receiver enable bit becomes the qualifier.

For data transmission, return to full speed works in conjunction with the loading of the serial-transmission buffer. This setup eliminates the need for manually exiting power-management mode prior to initiating a transmission.

- Selectable Clock Source

Crystal oscillators, of necessity, operate in the linear region which results in considerable power consumption. The ring oscillator, defined as operating in the 3–4-MHz range, is not accurate, but offers considerable power savings since it consists of a ring of digital gates. The DS87C530 not only uses the ring oscillator to speed emergence from Stop mode, but also uses it as the primary clock source under software control.

Selecting the ring oscillator as the clock source does not automatically disable the crystal oscillator. Instead, it reduces power consumption while leaving the option of returning to crystal operation if the system needs a precision time base. For instance, a switch to crystal would be necessary for serial port activity. When accurate timing is not necessary, the crystal oscillator can be shut down, leaving the ring oscillator running processes for a power savings of about 25 mA.

- Band-Gap Control

The DS87C530 uses a band-gap reference for monitoring V_{CC} to detect a power failure. While in Stop mode, the band gap can be a significant current burden in comparison to quiescent CMOS circuitry. Current drops from 80 µA to 1 µA if the band gap is not enabled. Although current savings are considerable, should a power failure occur, there is a distinct possibility of losing control of the system.

- Real-Time Clock

The DS87C530 incorporates a real-time clock and alarm that operates using its own 32.768-kHz crystal. When V_{CC} is off, the RTC keeps time by drawing power from an external back-up power source (usually a lithium cell or supercap).

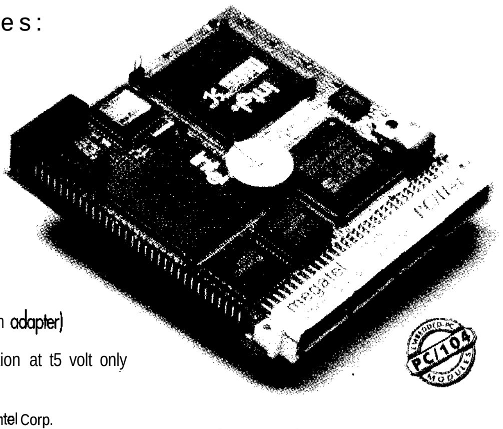
The RTC counts subseconds, seconds, minutes, hours, days of week, and weeks. Two total-day count registers are also included. Access to the RTC is via the DS87C530's special-function registers (SFRs). An alarm function generates an interrupt when the RTC values match selected alarm-register values. Since the RTC has its own crystal time base, it operates even when the processor is in Stop mode. This capability qualifies the RTC as a component of the DS87C530's power-management structure.

An alarm can be set to occur on a match with any or all alarm registers. Alternatively, it can be programmed for intervals of subseconds, seconds, minutes, or hours. Using the RTC alarms provides an on-chip means of taking the DS87C530 out of Stop mode

A UNIVERSAL COMPUTING ENGINE

The seemingly contradictory requirements of a general-purpose single-board computer and a data-collection system with full power management are brought together in the prototype system shown schematically in Figure 1.

The system consists of the processing core made up of the DS87C530, 33-MHz crystal, 74ACQ373 address



latch, and a 32-KB HM62832H 35-m RAM \PSEN and \RD are logically ORed using a 74AC08 gate so the RAM is accessible for both program and data memory.

A DS1210 provides RAM nonvolatility. Back-up power to the external RAM and the DS87C530's internal RAM and RTC are supplied by a BR1225 lithium cell. The master chip select to the RAM is sourced by the DS87C530's ALE signal. This connection ensures that RAM is never selected when the DS87C530 is in Stop mode and provides some interesting flexibility in how this RAM is mapped.

Using ALE as the RAM chip select enables the RAM whenever the processor is emitting a valid address. Although enabled, the RAM performs no action unless \PSEN, \RD, or \WR is asserted.

Memory access is resolved by the way the DS87C530 handles its external memory read and write strobes. For development, I am running the system under control of the Dunfield kernel in the controller's internal EPROM. If the application was executing from internal EPROM, using ALE as a chip select would be inadvisable since enabling the RAM when access was not necessary would increase consumption.

When the DS87C530 accesses internal memory-either EPROM or RAM-convention dictates that corresponding external access is blocked. That is, the controller is operating in single-chip mode and the port pins reflect the status of their respective SFRs. Since in my scheme, the external RAM is enabled for all memory accesses and, as a result, appears in both the lower and upper 32-KB blocks, access is controlled by how the DS87C530 partitions its internal memory resources. The DS87C530 has an SFR that provides the needed flexibility.

The ROMSIZE SFR selects the maximum on-chip decoded address for ROM. Increments of 0, 1, 2, 4, S, and 16 KB (default) may be selected. To access the external program RAM from the lower 32 KB, set this value to 2 KB, which enables external program access from 0800h upward.

Normally, this is not a problem since external program memory is defined to start higher than 2 KB to leave room for the system data area that is positioned at 0. In that case, moving the maximum on-chip EPROM higher gains space for additional EPROM-resident functions. This space is well used as the holding area for a ROMed Micro-C library.

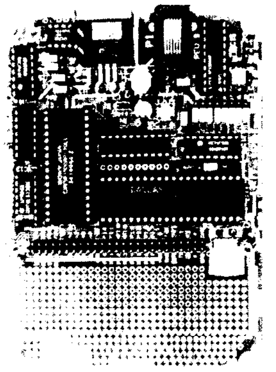
The 0-KB setting places the external program access at 0, the default reset vector. This setting is useful for downloading and testing the final version of the production code prior to burning the DS87C530 EPROM. For burn in, the kernel is invoked to load the program into low RAM (which functions as data memory) prior to switching itself out and reconfiguring the external RAM as program memory.

You may recall that the DS87C-530 has 1 KB of built-in data memory that also starts at 0. Similar to the internal program memory, access to the internal 1-KB RAM inhibits external access. This RAM can be enabled and disabled under software control, so the external data RAM appears in either the lower or the upper 32-KB area.

SYSTEM I/O

Although a number of elements characterize an embedded computer, most engineers agree that the most important is I/O. However, given the specialized nature of embedded applications, there is little agreement about what form this I/O takes. Multichannel analog I/O, digital I/O, high-current Darlington drivers, optically isolated inputs, and RS-232 or RS-485 network I/O are common methods of interfacing to external devices.

The system shown provides RS-232 communications since this link is required for the development PC to take control of the resident kernel. Keeping with the ideal of micropower operation, the system uses Linear Technology's LT1 180 RS-232 transceiver. An \OFF pin disables the built-in charge pump, tristates the communication pins, and places the chip into a 1-µA standby mode.



❑ ***Although a number of elements characterize an embedded computer, most engineers agree that the most important is I/O.***

Given the variety of I/O devices, rather than satisfy a small percentage of applications by imposing my view of what is useful, my system possesses no I/O beyond that of the processor. Obviously, this limitation renders the system next to useless. Expansion comes in the form of several identical 16-pin dual-row headers that carry all of the DS87C530's I/O lines not used for memory access and its power.

At a slight increase in material cost, this approach yields ultimate flexibility. All signals can be used for general I/O and some have additional alternate functions. A heavily interrupt-driven system can take advantage of the interrupt capability of the many I/O pins. Another application might use the lines for precision timing by having the relevant signals function as a high-resolution timer-capture system. With a couple of RS-485 transceivers, the system can function as a dedicated communications controller.

A wide range of peripheral chips are available with a synchronous serial interface. For the most part, these are I^{2}C and Microwire devices. I^{2}C lets you string a large number of functions using just two I/O pins. Microwire (or one of its many variants) defines a 3- or 4-wire interface that is typically much faster than I^{2}C.

A serial peripheral interface offers some important advantages. The expansion port is much less expensive since a small number of pins is required. More importantly, timing entanglements (if any) can easily be resolved since the exchange is performed entirely under software control. As well, many devices are available with extremely low-power standby modes for applications where low-power operation is desirable.

On the other hand, a 33-MHz microcontroller bus makes for some shrinking timing parameters, many of which fall well outside the operating limits of most peripheral chips. Even using stretch cycles, the setup and hold times can only be extended so much.

MICROPOWER PERIPHERALS

A number of micropower peripheral are suitable for the system I'm describing. The serial MAX1 86 is an 8-channel, 12-bit ADC with built-in 4.096-V reference. A software-invoked shutdown mode drops power consumption to 2 µA, bringing the MAX186 within system power-management criteria.

The familiar DS1620 with 1-µA standby current provides direct temperature conversion with serial output. The DS1620, although often used as a thermometer, is actually a thermostat with EEPROM configuration registers. If thermostatic operation is not required, the EEPROM registers can be used as general nonvolatile storage.

Since the LTC1392 is brand new, I'll elaborate more fully on this part. It is truly a micropower device using 350 µA active and 0.2 µA in standby mode. The LTC 1392 combines several very useful functions previously unavailable on the same chip.

Like the DS1620, the LTC1392 has a serial 3- or 4-wire interface and built-in temperature sensor. Also included is a differential rail-to-rail common-mode voltage input, and V_{CC} power monitor. Figure 2 depicts the functional blocks contained within the LTC1392.

Temperature conversion is provided in a IO-bit, unipolar output. Accuracy is specified over the –40–85°C operating range, although the converter offers readings beyond this range. The V_{CC} reading is also provided in a IO-bit, unipolar format. The conversion range extends from 2.42 to 7.255 V with a recommended range of 4.5-6 V.

A differential input voltage can be measured through the $+V_{IN}$ and $-V_{IN}$ pins. An input range of 0.5 V or 1 V is available for differential voltage measurements with resolutions of 7 or 8 bits, respectively. The rail-to-rail characteristic of the differential inputs indicates the converter is suitable for measuring current with the addition of a small sense resistor.

There's much more that could be said on the subject of hardware for low-power applications than I'm able to cover in this overview. It's a subtle topic where seemingly trivial decisions can ultimately make or break the power budget.

Perfecting the hardware is only the starting point. Now your software not only has to render the application, but must also integrate it into an overall power-management strategy.

That's next month's story. ❑

John Dybowski is an engineer involved in the design and manufacture of embedded controllers and communications equipment with a special focus on portable and battery-operated instruments. He is also owner of Mid-Tech Computing Devices. John may be reached at (203) 684-2442 or at john.dybowski@circellar.com.

CONTACTS

DS87C530 processor and DS1620
digital thermometer
Dallas Semiconductor
(214) 450-0448
Fax: (214) 450-0470

LTC 1392 data acquisition system
Linear Technology
1630 McCarthy Blvd.
Milpitas, CA 95035
(408) 432-1900
Fax: (408) 434-0507

MAX186 12-bit, g-channel
micropower ADC
Maxim Integrated Products
120 San Gabriel Dr.
Sunnyvale, CA 94086
(408) 737-7600
Fax: (408) 737-7194

I R S

425 Very Useful
426 Moderately Useful
427 Not Useful

CONNECTIME conducted by Ken Davidson

The Circuit Cellar BBS
300/1200/2400/9600/14.4k bps
24 hours/7 days a week
(203) **871-1988—Four** incoming lines
Internet E-mail: **sysop@circellar.com**

We're going to start out this month with a thread based almost entirely on opinion and past experience. We discuss the question: is it worth it to patent your latest invenfion? Please keep in mind when you're reading the messages that no participant in the thread is a lawyer. All opinions should be regarded as just that-opinions. A/ways consult with an attorney when considering applying for a patent.

In the next thread, we explore a number of options for sensing when AC current is flowing. By not requiring a quantitative measurement, the solution is kept straightforward.

Finally, we look at the analysis of what appears on the surface to be a relatively simple circuit. Computers come to the rescue again.

Patents-worth it?

Msg#:10665
From: Greg Bell To: All Users

Given the amount of ingenuity that comes from *Circuit Cellar INK* authors and readers, I thought this would be a good place to ask: Does anybody go to the trouble to patent their stuff? How worthwhile has this proven? Do you enlist the help of patent lawyers or do everything yourself?

Don Lancaster has written volumes trying to convince people that patents are a big waste of time and money. He claims all they really do is give you the right to sue if some company steals and markets your idea, and as a "product developer" you don't have the resources to fight some big company in court.

Opinions?

Msg#:12216
From: Dale Nassar To: Greg Bell

My advice is, if it's complex, forget it!

I patented a device for a company I worked for several years ago. The patent was granted in 1987. They keep sending bills for "maintenance fees" and such.

The patent was rather technical. The title is "Multiplexed Dual-Tone Multifrequency (DTME) Encoding/ Decoding System For Remote Control Applications," Patent #4,670,875. We were hoping to sell rights to someone like AT&T, maybe.

The purpose was to produce a system to transmit secure control signals over a noisy audio channel such as a crowded CB channel. The system used DTMF signals and allowed more than the standard 16-key limit and multiple buttons could be pressed simultaneously-all while preserving (actually improving) the DTMF reliability.

The patent lawyer was a very bright fellow with a background in EE. So I assumed I could write a description of standard DTMF operation with a detailed description of my modification-WRONG!

I spent over 100 hours in his office going over every word of his and my writings (he's paid by the hour!) until it was "just right for DC." Also, for reasons I don't really remember, it was necessary to publish the complete and detailed schematic in the text-down to the component tolerances and associated formulas. Now tell me, how does this protect you? Anyone can now go to the local university library, get a copy of the schematic and circuit description, and build the device-who would know if it was being sold?

Further, any patent with any degree of complexity could be infinitesimally altered, repatented, and produced (I would guess) all from reference to the original.

I also learned that the device does not have to even exist to be patented.

So I guess I agree with Don Lancaster.

Msg#:13100
From: Robert Lunn To: Greg Bell

For ten years I have worked for a small company that has successfully registered about a dozen patents over that period. I have been personally involved in about half of these.

1. Do not even consider doing it without a patent attorney, and a *good* patent attorney at that.
2. The only part of a patent that really matters are the claims. It takes a technically sophisticated poet to generate good claims.
3. Successfully lodging a patent in all significant countries (U.S., Europe, Australia, etc.] will easily cost $50,000.
4. In addition, there are substantial annual fees to maintain the patent.
5. A patent, once granted, is public domain. A patent simply tells your competitors what they *cannot* do.

CONNECTIME

The first one would be a low-voltage (say 6 V) AC relay. You could run your load current through the coil in a series connection (not the usual parallel feed). If your load never gets very heavy, the relay alone might do just fine.

If the load current could potentially be much higher than the relay activation current, you should put a pair of series-connected zener diodes (3- or 5-W types, 6.8 V) over the relay coil. The two diodes would be wired cathode to cathode and thereby produce a bipolar clamp of about 7.5 V peak. With 5-W zener diodes, you could handle a maximum of about 1 A.

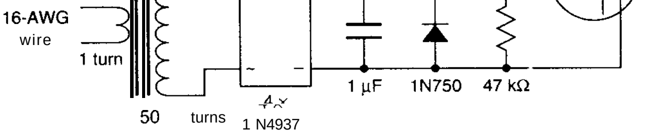
Not enough? Thought so! Then you need a current transformer. And once you have that, you can still use the relay and clamp on the secondary. Select the turns ratio to provide the sensitivity and range that you need. But you probably cannot find commercial current transformers quite suitable for the low burden rating the relay **would** represent. And the price of the usual 5-A secondary type current transformers may also be a little more than what you seem to be indicating.

Wind your own? It's no big deal if you have the materials around, but may be more than most people are willing to do. So, here comes the other suggestion.

Get one of the toroidal current-sensing coils that are commonly used in the switching power supplies. They are designed to be operated with a single-turn primary and have a burden rating of a fraction of a watt. They saturate at low frequencies, but that is fine for the on/off detection.

On the secondary of that transformer, put a high-voltage (600 V or more) fast-diode bridge. Then put a film capacitor of any reasonable value (such as 1 µF) from the positive to the negative end of the bridge. Add a zener diode clamp and a load resistor that allows the charge to decay when there is no primary current. Then use whatever electronic means you want, such as a simple FET, an op-amp, or a comparator, for getting up to the power levels you need. Remember that the current transformer of this type cannot put out any real power.

Let's see if I can make the idea a little more readable by some graphics.



For the ferrite core, you can probably find suitable stuff from Radio Shack. If nothing else, they carry some openable U-I cores in a plastic frame. They are intended for noise filtering, but should do fine in the application I describe here.

During the 60-Hz cycle, the transformer secondary puts out high-voltage spikes every time the magnetic flux reverses. The spikes are short if the core size is not excessive and the energy there is minimal. The 1N750 zener diode should be able to handle the low average power. You might want to select a more powerful zener diode if you end up using any bigger core sizes.

The sensitivity can bc trimmed by the number of turns on the secondary. I think 50 turns should be suitable for the currents that typically would be used with 16-AWG primary wire. I must present a disclaimer: I have not built this exact circuit for this exact application, so some experimentation with component values may be necessary.

Msg#:16164
From: George Novacek To: Don Meyer

There are a number of manufacturers making and selling current transformers. All you do is run a wire through the transformer which looks like a toroid. They come in many different sizes for wide range of currents, and many are PCB mounted. On the secondary, you have a load which converts the primary current to a scaled secondary voltage. You rectify this and follow with threshold detectors or a readout which is scaled to the current or whatever you want. The current transformer manufacturers may have some application notes. A lot of articles have been written on the subject, including this BBS.

If the current is DC as opposed to AC, your best choice would be Hall-effect sensors. Here are just a few suppliers of current transformers and Hall-effect current sensors:

Ohio Semitronics, Inc.
(614) 777-1005
Fax: (614) 777-4511

Coilcraft
(800) 322-2645
Fax: (708) 639- 1469

LEM USA, Inc.
(414) 353-0711
Fax: (414) 353-0733

If you want to experiment a bit, you can wind one yourself using a small toroid core. It's too not difficult. We just went through the process because we could not find exactly what we needed off the shelf, but I would not do it otherwise.

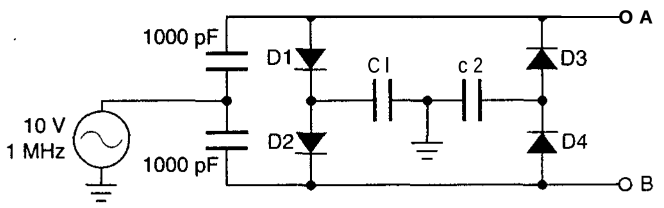
If you are talking just a few amps, you can use a small resistor. Add an amp if you want to keep the resistor very small and minimize heat dissipation. Use an optocoupler or an isolation amplifier (Burr-Brown) and power it from the 120-V line-you need just a few milliamps. This could be done with a bleeder capacitor. When you work with 120 V, you have to consider safety!

Also, there are current-sensitive relays which have fairly small voltage drop. You put them in series with the load. You can also use a 5-V relay in combination with a small resistor in-line with the load or a bipolar surge arrestor which will provide relatively constant voltage drop and a corresponding voltage-sensitive relay (make sure it is an AC type or it will chatter). Such solutions are countless, although they must be tailored to the load. They are inexpensive and easy to implement.

Capacitive tilt sensor

Msg#: 5823
From: Rick Vitucci To: All Users

I have been developing a tilt sensor based on a capacitive transducer. The circuit below is from a NASA Tech Brief. Cl and C2 represent the transducer (they are both variable caps between 4 and 30 pF). When Cl equals C2, the DC voltage between A and B is zero. When Cl and C2 are different, the output has a DC level (somewhere between -1.5 and +1.5 V). Normally, there is a low-pass filter to cut the AC at the output, and I get a clean DC level which can be displayed on a scope.



All diodes are matched

At first, I thought I knew how the circuit worked. It looks simple. Then I started analyzing it on paper, and it turned into a real mess. I am looking for an expression for the output voltage AB and some kind of explanation of the charge on all caps as the input voltage swings through a cycle.

Any help is greatly appreciated.

Msg#: 5885
From: James Meyer To: Rick Vitucci

I put the circuit in PSpice and played around with the parts values enough to get a handle on its operation. The output voltages when simulated are close to what you see in “real life.” I used a model of a 1N4148 for the diodes.

The way I make it, the circuit is basically a voltage divider. The output voltage is derived from the division of the generator voltage across each of the 1000-pF caps and the variables Cl and C2. The diodes are switches that connect C l to the upper 1000-pF cap when the generator voltage is at one polarity and to the lower 1000-pF cap when the voltage is at the other polarity. C2 is connected similarly, except on opposite generator polarities.

Or, put in other words, when Cl is larger in value than C2, point A sees the upper 1000-pF cap lightly loaded on positive generator voltages and more heavily loaded on negative voltages, so the average voltage at that point goes positive. The reverse is true at B, and it goes negative.

Ten volts would divide between 1000 pF and 30 pF to give about 9.67 V and 4 pF to give 9.99 V. That’s a difference of 0.32 V. Double that for the differential action of the output and I figure the output voltage should be about 0.64 V when the sensor is at one extreme.

The PSpice simulation gave me a differential output voltage about twice what my figures above show and much closer to what you found with the “real” circuit. Perhaps it’s due to my use of 10 V without regard to whether it’s RMS or peak or perhaps even peak to peak.

As a test of my theory, I reduced the value of the 1000 pF caps to 100 pF each and the differential output voltage went *up . when the sensor wasunbalanced.

I suspect a precise mathematical expression for the output voltage would be hairy, but it looks like it’s a nice linear function of the differential capacitance of the sensor [as long as the sensor caps are at least 10 times ‘smaller’ than the two coupling caps] and directly proportional to the input voltage.

Msg#: 6130
From: Rick Vitucci To: James Meyer

James, thanks for being so enthusiastic and doing a simulation. I had a feeling that someone wouldn’t be able to resist the temptation. I have to get myself a good working version of PSpice, and take some time to learn it.

The sensor response is pretty much linear and very sensitive for angles less than 15”. After that, it loses sensitivity and isn’t good for much, except giving the direction of tilt. I haven’t had a chance to figure out why. Maybe it’s due to the geometry of my plates. I also have a bit of a hysteresis problem due to surface tension.

CONNECTIME

Msg#: 6321
From: James Meyer To: Rick Vitucci

PSpice is great for me because it lets me see pictures instead of words. I understand things better that way. It also gives me feedback. If I think something works a particular way, I'll predict what should happen when I change a component. If I guess right, PSpice agrees. Of course, if I'm off base, it lets me know that too-really quickly. S-l

Msg#: 6716
From: Pellervo Kaskinen To: Rick Vitucci

The circuit you describe uses something known as ring modulator. I was sure I could locate the analysis of its functioning in any number of electronics text books. What a surprise! Only so much verbiage and no analysis. Besides what Jim Meyer has done, it may be that one of my professors may have been the only one persistent enough to dig in....

I recall that it was covered in an Applied Electronics course, but I do not have the notes with me to be able to check. And anyway, it is only important for my own peace of mind.

Mini Circuits sells a line of double-balanced modulators that, to my understanding, are based on the ring modulator. The important aspect for that application is that you have good isolation between all three ports [e.g., antenna, local oscillator, and the IF port). That makes it easier to keep signals clean and EMI minimized. It produces a DSB signal with 40 to 50 dB carrier attenuation. According to an old *Radio Amateur's Handbook,* the beam deflection tube can achieve 60 dB carrier attenuations. But we know what size that thing is [if we know anything at all about vacuum tubes, that is). :-)

These are just side notes. There is nothing much of substance I could add to Jim's analysis, except maybe the fact that the signal amplitude range is limited by the normal silicon diode clamping effect (logarithmic characteristic). You can expect to get less than 1 V RMS in a fairly linear fashion.

This may or may not be the reason for your loss of sensitivity at larger tilts. The reason could also be something inherent inside the sensor. The least likely thing would be the trigonometric relationships. At 15°, you should not see too much of that effect. If your results are good up to 60" or similar, then I would think the opposite to be true.

Msg#: 6871
From: James Meyer To: Pellervo Kaskinen

The diodes are used in a switching mode and so the forward-biased E-versus-I characteristics do not limit the output voltage in this application. The output voltage when the sensor caps are unbalanced continues to rise as the applied excitation voltage rises. If the coupling capacitors and sensor capacitors are all made 100 times larger in value so the capacitance associated with the diodes becomes insignificant in proportion to them, then the output voltage is quite linearly related to the excitation voltage to large values. Putting 100 V in will get you 20 V out.

PK> This may or may not be the reason for your loss
PK> of sensitivity at larger tilts.

PSpice doesn't think so. 8-}

PK> The reason could also bc something inherent
PK> inside the sensor.

That idea gets my vote. I suspect something in the mechanical design of the sensor limits the range of delta capacitance. That should be easy to check with a capacitance bridge or meter.

By the way, and for the benefit of lurkers, PSpice is available right here in the files area.

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 (203) 871-1988. Set your modem for 8 data bits, 1 stop bit, no parity, and 300, 1200, 2400, 9600, or 14.4k bps. For information on obtaining article software through the Internet, send E-mail to info@circellar.com.

ARTICLE SOFTWARE

Software for the articles in this and past issues of *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 (203) 87.52199. Be sure to specify the issue number of each disk you order. Please add $3 for shipping outside the U.S.

IRS

428 Very Useful 429 Moderately Useful 430 Not Useful

STEVE'S OWN INK

Under the Covers

If you were around for the first issue of *INK, you'll* remember I titled that issue "Inside the Box Still Counts." As a new magazine, evolving at a time when other mainstream magazines seemed to becoming "plug-n-go" business software reviewers, I wanted to remind a special audience that somewhere, under all that business hype, was still a piece of electronic hardware.

I won't lie to you. I view things like an engineer. I look at the problem, try to understand the goals, and then map out a course to achieve results in the most cost-effective manner. When someone starts telling me I have to be politically correct (technically speaking, that is), they've suppressed alternatives. When an engineer is told that all design situations must incorporate a prescribed hardware solution, they've eliminated invention. That's how I feel when someone demands I use a PC to do something better suited to a single-chip 8751!

The dilemma for me has always been a tradeoff between technically correct riches or parochial happiness. For those of you who think I'm anti-PC, guess again. Consider the Circuit Cellar record when it comes to PCs. If I remember correctly, the first PC-clone on the market was Columbia Data Systems. The second PC-clone, the MPX-16, was from Circuit Cellar. A few years later when single-board industrial PCs with passive backplanes were introduced, Circuit Cellar brought you the OEM-286. Somewhere in between, I introduced Trumpcard, one of the first coprocessor cards (28000) for the PC.

Successful embedded control engineers and software designers rely on their inventiveness. They know a true embedded control design is a cost compromise of software development, hardware duplication, raw computing power, and accessible I/O. They *also* know that nothing is static. For years, the cost of adding analog I/O to a PC was outrageous. Then, for a while, development tools cost a fortune. Now we're seeing cheap tools and more generic controller architectures.

So, with certain embarrassment, I admit that while I might have been in the revolution and even lit a few cannon fuses, publishing is more like drawing the battle plan than participating in the war. We started a magazine to promote the logical correctness of using embedded control, not the correctness of a specific controller. When a PC is packaged and presented to fit that logical compromise equation, however, it cannot be ignored.

Technology and applications have evolved to the point now where there can be as much of an embedded match using a '386 or Pentium PC as there is for a PIC processor in a credit card reader. I don't have to eat my words to say that using an embedded PC makes all the sense in the world. If it looks like a duck, quacks like a duck.... Choosing an embedded PC controller is now more likely an intelligent preference than a political gesture.

As with other earthshaking technical subjects, *INK* plans broad and intense coverage. Starting next month, we are presenting a special quarterly section called *Embedded PC.* Like the *Home Automation and Building Control* special sections, *Embedded PC* starts at *32* pages each time. We trust it will grow quickly.

Like the rest of *INK, Embedded PC* is application oriented. It presents hands-on software and hardware development topics. New columnists will be joining us to present their solutions to real-world hands-on control problems.

The embedded FC has come of age and should be a logical consideration. When I think of using it, I can wipe away past horror stories about 75-lb desktop PCs with rubber-cushion-mounted hard drives, 300-W power supplies, heat-pump-processor coolers, and octopus I/O wiring, all jammed into a 5-lb box. Now, the same mess is a palm-sized stack they'll soon want you to jam into a thimble.

Steve