

EMBEDDED PC  
MONTHLY SECTION

# CIRCUIT CELLAR<sup>®</sup> INK<sup>®</sup>

THE COMPUTER APPLICATIONS JOURNAL

#94 MAY 1998

## DATA ACQUISITION

The Black Art of Analog Design

Antialiasing—Just a Matter of Know-How

Selecting the Right Position Transducer

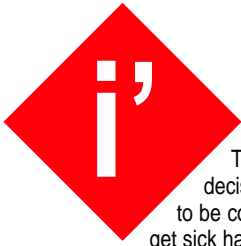
Creating Virtual Tools



\$3.95 U.S.  
\$4.95 Canada

# TASK MANAGER

## A Race Well Run



I'll never forget my first road race that cold Thanksgiving Day 1996. I had to make so many decisions that morning. What to wear? I didn't want to be cold, but it's terrible to overheat. If I ate, would I get sick halfway through the course? If I didn't, would I have the energy to finish? Where should I line up? If I was too close to the front, would I get bowled over by speed-machines? Where was too close? All the contestants looked like they ran hundreds of miles, did weights, and then biked for leisure.

Looking back on it now, I know I just had a bad case of nerves. I had been running for almost a year. I was ready, and besides, 5 miles isn't far for someone running 3-6 miles per morning. It was just the panic of bringing myself up to the line and having to compete with others who might be better. However, despite knowing this, every time I go to race, that knot in my stomach comes back.

In truth, my reactions are pretty typical. Nearly everybody, when faced with the opportunity/risk of competing with someone else, feels a little intimidated. Whether it be a road race, a business proposal, a job interview, or a Design98 submission, you don't know how you're going to fare. And, it's nerve-racking.

As I surveyed the entries when they came in, I noticed this same anxiety. International competitors wrote cover letters apologizing for their English, one fellow called up panicking that he'd left his name in the project in three places, while others wanted to send in countless revisions to this or that paragraph.

What does all this anxious energy indicate? That a lot of you poured a whole lot of effort into your projects, that you were proud of your designs, that you really wanted them to win. You didn't want to be eliminated because of your English, the inclusion of your name, a lack of a photo, whatever.

As I watched you turn the first corner and head up the mile-long hill, you were an impressive pack. It was the largest showing we'd ever had. All populated continents were represented, with 16% of the entries coming from Europe, 8% from South America, 8% from Canada, and 9% from Australia/New Zealand. We even received 2 from Asia and 1 from Africa. And, in the final analysis, 60% of the winners are non-American. Perhaps coincidentally, 60% of Microchip's sales are from international markets.

As you hustled your way across the finish line, I saw many good solid finishers who, like myself, didn't make it to the top ten, but are nonetheless good designers. (I'm trying to get manuscripts from all the competitors. If we can't fit them in *INK*, we can cover the designs on our Web site.)

And, as you now enter your postrace moments, I hope you share many of the same feelings that I experience. I look on those top winners and marvel that anyone's legs can cover ground quite as quickly as they did. Then, I look around me and see the many others that finished at about the same time as I did and I realize that yet again, I finished. I stepped up to the mark, I ran the race, I finished the course. I accomplished what tens of millions of other people would never dream of doing—I went above and beyond the call of duty. I entered a contest and pitted my own skills against another's.

When you do that and you feel the pride well up within you, then you know you really won, regardless of what the judges or the clock have to say. And, with that feeling you know without a shadow of doubt that you'll do it again, only next time, you'll do it just a little better and just a little smarter.

janice.hughes@circuitcellar.com

# CIRCUIT CELLAR<sup>®</sup> INK<sup>®</sup>

THE COMPUTER APPLICATIONS JOURNAL

## EDITORIAL DIRECTOR/PUBLISHER

Steve Ciarcia

## ASSOCIATE PUBLISHER

Sue (Hodge) Skolnick

## EDITOR-IN-CHIEF

Ken Davidson

## CIRCULATION MANAGER

Rose Mansella

## MANAGING EDITOR

Janice Hughes

## BUSINESS MANAGER

Jeannette Walters

## TECHNICAL EDITOR

Elizabeth Laurençot

## ART DIRECTOR

KC Zienka

## WEST COAST EDITOR

Tom Cantrell

## ENGINEERING STAFF

Jeff Bachiochi

## CONTRIBUTING EDITORS

Ingo Cyliax  
Fred Eady  
Rick Lehrbaum

## PRODUCTION STAFF

John Gorsky  
James Soussounis

## NEW PRODUCTS EDITOR

Harv Weiner

Cover photograph Ron Meadows – Meadows Marketing

PRINTED IN THE UNITED STATES

## ADVERTISING

### ADVERTISING SALES REPRESENTATIVE

Bobbi Yush  
(860) 872-3064

Fax: (860) 871-0411  
E-mail: bobbi.yush@circuitcellar.com

### ADVERTISING COORDINATOR

Valerie Luster  
(860) 875-2199

Fax: (860) 871-0411  
E-mail: val.luster@circuitcellar.com

## CONTACTING CIRCUIT CELLAR INK

### SUBSCRIPTIONS:

INFORMATION: [www.circuitcellar.com](http://www.circuitcellar.com) or [subscribe@circuitcellar.com](mailto:subscribe@circuitcellar.com)  
TO SUBSCRIBE: (800) 269-6301 or via our editorial offices: (860) 875-2199

### GENERAL INFORMATION:

TELEPHONE: (860) 875-2199 FAX: (860) 871-0411  
INTERNET: [info@circuitcellar.com](mailto:info@circuitcellar.com), [editor@circuitcellar.com](mailto:editor@circuitcellar.com), or [www.circuitcellar.com](http://www.circuitcellar.com)  
EDITORIAL OFFICES: Editor, Circuit Cellar INK, 4 Park St., Vernon, CT 06066

### AUTHOR CONTACT:

E-MAIL: Author addresses (when available) included at the end of each article.  
ARTICLE FILES: [ftp.circuitcellar.com](ftp://ftp.circuitcellar.com)

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




CIRCUIT CELLAR INK<sup>®</sup>, THE COMPUTER APPLICATIONS JOURNAL (ISSN 0896-8985) is published monthly by Circuit Cellar Incorporated, 4 Park Street, Suite 20, Vernon, CT 06066 (860) 875-2751. Periodical rates paid at Vernon, CT and additional offices. One-year (12 issues) subscription rate USA and possessions \$21.95, Canada/Mexico \$31.95, all other countries \$49.95. Two-year (24 issues) subscription rate USA and possessions \$39, Canada/Mexico \$55, all other countries \$85. All subscription orders payable in U.S. funds only via VISA, MasterCard, international postal money order, or check drawn on U.S. bank.

Direct subscription orders and subscription-related questions to Circuit Cellar INK Subscriptions, P.O. Box 698, Holmes, PA 19043-9613 or call (800) 269-6301.

Postmaster: Send address changes to Circuit Cellar INK, Circulation Dept., P.O. Box 698, Holmes, PA 19043-9613.

Circuit Cellar INK<sup>®</sup> 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<sup>®</sup> disclaims any responsibility for the safe and proper function of reader-assembled projects based upon or from plans, descriptions, or information published in Circuit Cellar INK<sup>®</sup>.

Entire contents copyright © 1998 by Circuit Cellar Incorporated. All rights reserved. Circuit Cellar INK is a registered trademark of Circuit Cellar Inc. Reproduction of this publication in whole or in part without written consent from Circuit Cellar Inc. is prohibited.

- 12 Practical Analog Design**  
*Bob Perrin*
- 26 Antialiasing Techniques**  
*Mike Zerkus*
- 34 Selecting Position Transducers**  
*Tom Anderson*
- 60 PostScript Flutterwumpers**  
*Don Lancaster*
- 66**  **MicroSeries**  
EMI Gone Technical  
Part 4: Transient-Suppression Design Technique  
*Joe DiBartolomeo*
- 72**  **From the Bench**  
Rebirth of the Z8  
Part 2: Let Your Micro Answer All  
*Jeff Bachiochi*
- 78**  **Silicon Update**  
FPGA Tool Time  
*Tom Cantrell*

**Task Manager** 2  
Janice Hughes  
A Race Well Run

**Reader I/O** 6

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

**Advertiser's Index** 65

**Priority Interrupt** 96  
Steve Ciarcia  
Design98—A  
Marketer's PICnic

# INSIDE ISSUE 94

## EMBEDDED PC

- 41 Nouveau PC**  
*edited by Harv Weiner*
- 46** RPC **Real-Time PC**  
Graphical User Interfaces in RTOS  
*Ingo Cyliax*
- 54** APC **Applied PCs**  
A New View  
Part 1: Virtual Instrumentation  
*Fred Eady*

[www.circuitcellar.com](http://www.circuitcellar.com)

# READER I/O

## NO NEED FOR DOOM

Many traffic lights operate on processors that track the day of the week based on the date. This gives them the ability to run weekend and "normal" schedules.

In the year 2000, a lot of these have clock chips with only two digits of year, and the software will believe that it's 1900 (see Scott Lehrbuam's "Year 2000 and Embedded PCs," *INK* 90, for a clear exposition of the problem).

The fix, however, is simple. Instead of trying to work around a two-digit year, all you have to do is supply a year to the controller which is compatible for the day of the week.

Secondly, the new standard for all traffic lights, digital road signs, traffic sensors, and so forth is called NTCIP ([www.ntcip.org](http://www.ntcip.org)). All this equipment is to be controlled via TCP/IP and SNMP. Therefore, the fix will be easy to implement.

**Rod Price**

[gacoac@netdepot.com](mailto:gacoac@netdepot.com)

## INTEL INFO

"MicroBot" (*INK* 92) uses an Intel D8479H controller, which is considered an end-of-life part ([developer.intel.com/design/specenvn/seupdt2/index.htm](http://developer.intel.com/design/specenvn/seupdt2/index.htm)), and there is no direct replacement. The suggested upgrade path is the MCS-51 processor family, but it doesn't share the same instruction set and isn't pin-for-pin compatible.

Although the Intel Publication #270646-005 (Embedded Microcontrollers) is mentioned, its current revision (-010) doesn't contain information on the 8749. Rather than calling the general switchboard, readers should contact Intel Customer Support at (800) 628-8686 (Fax: (916) 356-2892). The URL for information on Intel Microcontrollers is [developer.intel.com](http://developer.intel.com).

If anyone is interested in building MicroBot, they may be able to find 8749 parts from America II Corp. ([www.americaii.com](http://www.americaii.com), (800) 767-2637).

**Craig Miller**

[Craig\\_S\\_Miller@ccm.fm.intel.com](mailto:Craig_S_Miller@ccm.fm.intel.com)

# NEW PRODUCT NEWS

Edited by Harv Weiner

## CONFIGURABLE DEVELOPMENT SYSTEM

**HOT Works**, a Hardware Object Technology development system, bundles all the hardware and software you need to experiment with configurable computing. HOT Works includes the Xilinx XC6200 reconfigurable processing unit and the XC4000 FPGA. Their open architecture and dedicated microprocessor interface are ideal for rapid partial reconfiguration from within executable programs. The HOT Works PCI plug-in board provides a standard platform for developing applications using hardware objects (digital logic designs translated into static arrays).

The HOT Works package enables the hardware engineer to implement and test designs in real time. The HOT Works board lets the user examine the XC6200 family through real-time emulation. And via a high-level hardware-description language, hardware-object technology, and plug-in coprocessing board, the software engineer can use this development system for algorithm acceleration.

The package includes the Lola Programming System, which is a simple hardware-description language for synchronous digital circuits, and Xilinx's XACTStep-6000, which is a map, place, and route toolset. Also featured is the XC6200 VHDL Elaborator, a third-party tool for converting VHDL to EDIF, and the Hardware Object Technology Interface for inserting designs into executable C programs (enabling run-time reconfigurable computing). WebScope 6200, a Java tool for real-time design emulation using the PCI-XC6200 board, and design examples are included as well.

Together, the HOT Works PCI plug-in board and development software kit sell for **\$995**. The prototyping daughter-board is priced at **\$199**.

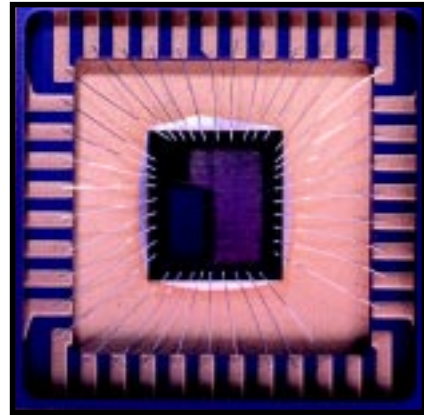
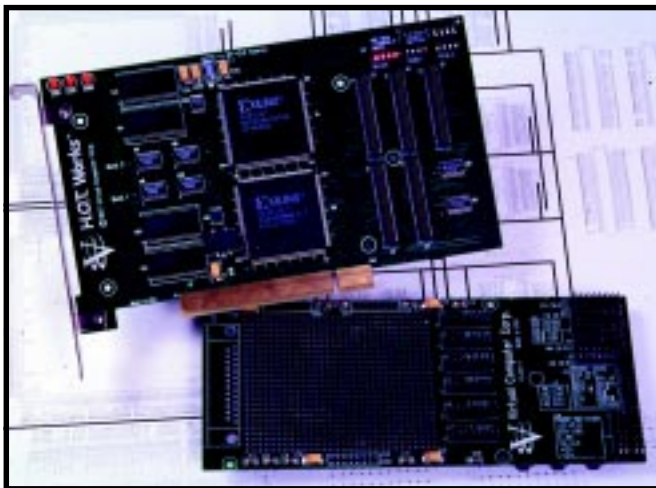
### Virtual Computer Corp.

6925 Canby Ave., Ste. 103 • Reseda, CA 91335

(818) 342-8294 • Fax: (818) 342-0240

[www.vcc.com](http://www.vcc.com)

#501



## SUPER-FAST 8-BIT MICROCONTROLLER

The **SX Series** microcontrollers from Scenix Semiconductor deliver up to 50-MIPS processing power at 50 MHz and suit many applications considered the realm of 16- and 32-bit MCUs, low-end DSPs, and custom ASIC designs. This performance lets designers use virtual peripherals—software implementations of functions previously requiring costly dedicated hardware. This code occupies a small part of the onboard 2K × 12-bit flash memory, requiring relatively few MCU resources for execution.

A four-stage (fetch, decode, execute, write back) pipeline executes one instruction per clock cycle, yielding a 20-ns instruction cycle at 50 MHz. A fast (12-ns access time) embedded flash program memory and a correspondingly fast SRAM register file give a deterministic interrupt response time of 60 ns for internal and 100 ns for external events.

The SX offers a 4-MHz ( $\pm 3\%$  accuracy) programmable oscillator, programmable three-level brown-out reset, and power-on reset. It has a watchdog timer with RC oscillator and multi-input wakeups, and an on-chip analog comparator can be used with hardwired components or software techniques to provide potentiometer or temperature-sensing capabilities. The SX architecture contains 43 instructions (33 are object-code compatible with PIC16C5x MCUs).

The first production SX Series MCUs—the SX18AC and SX28AC—are 18- or 28-pin package (DIP, SSOP, and SOIC), 2048 word ( $\times 12$ ) devices priced at \$3.49 in 1000 quantities.

### Scenix Semiconductor, Inc.

3140 De La Cruz Blvd., Ste. 200

Santa Clara, CA 95054

(408) 327-8888 • Fax: (408) 327-8880

[www.scenix.com](http://www.scenix.com)

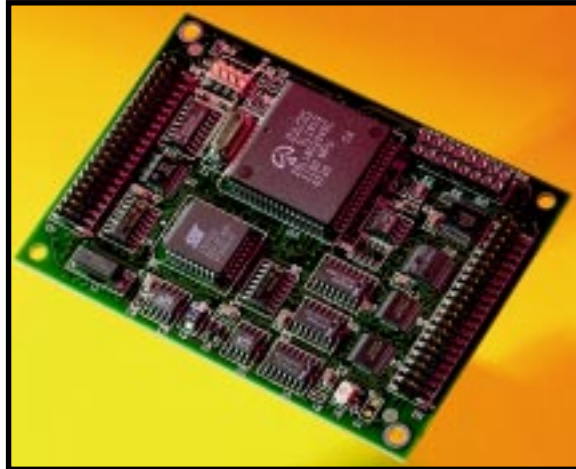
#502

# NEW PRODUCT NEWS

## LOW-POWER C-PROGRAMMABLE CONTROLLER

The **LP3100** low-power C-programmable controller from Z-World operates on 3.3 V and consumes only 16 mA. Its power subsystems are independent, so resources can be shut down to conserve power. This feature makes the LP3100 ideal for operation in mobile or remote installations, battery-powered embedded systems, and OEM applications. Measuring only 2.5" × 3.5" × 0.5", the LP3100 can be embedded into the tightest of spaces.

The LP3100 features 20 digital I/O lines, four channels of conditioned 12-bit analog input, two RS-232 serial channels, an RS-485 port, 512-KB flash memory, a real-time clock/calendar, and an LPBus expansion port. The unit operates on input power in the range of 3.5–24 VDC. The LPBus expansion port facilitates the addition of user-designed boards for direct use with the LP3100 controller.



This unit comes equipped with a sleep mode, so it can shut itself down to conserve power, reducing the required current to 200  $\mu$ A. Sleep mode is invoked by software, and the real-time clock can be used to wake up the LP3100 at a specified date or time.

The LP3100 is available with a development kit that includes a reference manual, serial cables, programming cable, wall power supply, development board, 2 × 20 LCD, and mounting plate.

Pricing for the unit starts at **\$119** in single quantities.

**Z-World**  
2900 Spafford St.  
Davis, CA 95616  
(530) 757-3737  
Fax: (530) 753-5141  
[www.zworld.com](http://www.zworld.com)

#503

# NEW PRODUCT NEWS

## MULTICHANNEL DATA-ACQUISITION CARD

The **T8ADH** provides eight channels of analog and digital I/O on a credit-card-sized expansion board. All eight channels are individually configurable as a 0-5-V 12-bit analog input, a digital input (with input latch), or a high-sink-current digital output. Digital output and analog input functions can coexist.

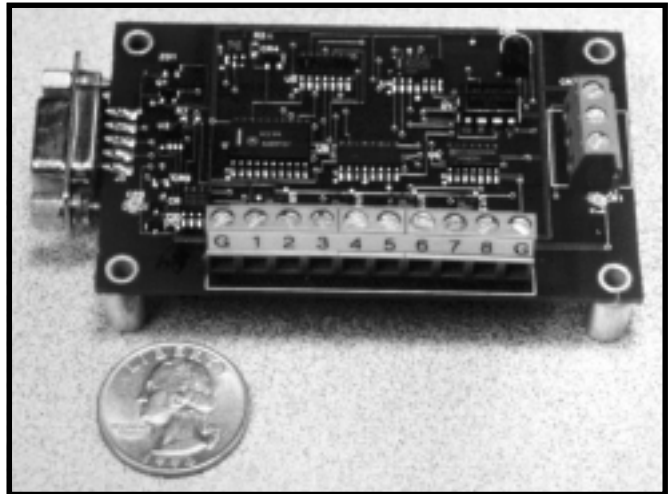
The T8ADH features five multidrop controllers (DS2407), which provide five unique 64-bit registration numbers (8-bit family code plus 48-bit serial number plus 8-bit CRC). These codes assure error-free selection and absolute identity of the device—no two parts are alike. This unique addressing means that I/O functions can be identified absolutely, virtually eliminating DIP-switch confusion.

The T8ADH has a built-in RS-232 to one-wire interface, enabling network expansion to drive up to 200 one-wire devices (e.g., temperature, pressure, force, humidity, pH, etc.) over 2000' of Cat-5 twisted pair cable. All necessary power is derived from the RS-232 port, and all data transfers are CRC 16 error checked.

An included DDE driver allows interface to most Windows applications. The T8ADH sells for **\$119.95**.

Point Six, Inc.  
138 E. Reynolds Rd., Ste. 201  
Lexington, KY 40517  
(606) 271-1744  
Fax: (606) 271-4695  
[www.pointsix.com](http://www.pointsix.com)

#504



# FEATURES

12

Practical Analog Design

26

Antialiasing Techniques

34

Selecting Position Transducers

60

PostScript Flutterwumpers

## FEATURE ARTICLE

Bob Perrin

# Practical Analog Design

Time for a refresher on op-amps? If so, here's the article for you. Bob covers op-amps from the basics of common op-amp configurations and modeling to the more advanced aspects of instrumentation amplifiers and complex circuit designs.



analog design is fraught with snares and pitfalls. In all the years I've been designing mixed-signal instrumentation, I've made my share of mistakes, but I've learned a bit, too. I want to share some practical circuit designs to make your experiences easier.

As you know, many texts cover op-amp modeling in detail. Why? Because op-amps aren't simple devices.

A typical SPICE model for a transistor has about 30 parameters to describe the transistor's behavior. An op-amp has 30 or more transistors. Imagine the interaction of all of the various transistor parameters.

### OP-AMP BASICS

While real-life op-amps are complex assemblies, we can still get a lot of mileage out of simple models. I'll give you a refresher course on common methods for predicting the behavior of op-amp circuits.

To begin, an op-amp is a high-gain differential amplifier. The voltage difference between the noninverting (+) and inverting (-) inputs is amplified by the open-loop gain,  $A_o$ .

The model shown in Figure 1 predicts op-amp behavior reasonably well for most circuits. There are of course



many second-order effects this model doesn't taken into account.

$A_o$  is the single most important parameter influencing op-amp behavior. The open-loop gain is generally  $10^6$  or  $10^7$  V/V at DC. In the most common op-amps (i.e., those that are internally compensated),  $A_o$  typically begins a 20-dB-per-decade rolloff starting at a few hertz. Figure 2 shows  $A_o$  as a function of frequency.

Internally compensated op-amps have a dominant pole at  $f_o$  and trade open-loop gain for stability. These are the easiest op-amps to design with because they're stable over the widest range of closed-loop gains and load impedances.

Noncompensated devices lack a dominant pole at  $f_o$ , enabling the amplifier to perform well at higher frequencies. However, the designer must ensure the system remains stable. Datasheets for noncompensated devices indicate the range of stable closed-loop gains.

Both compensated and noncompensated op-amps oscillate at some point when driving some types of reactive loads. A slightly capacitive load is usually the cause.

Noncompensated devices are most sensitive to load impedance. Unless you need good high-frequency characteristics, using compensated devices makes life much simpler.

When you're selecting an op-amp, especially for a high-speed application, it's imperative to know the behavior of  $A_o$ . Manufacturers talk about the gain bandwidth product (GBP), which is the product of  $A_o$  at unity gain (i.e., 0 dB or 1 V/V) and frequency  $f_t$ . The unit for the GBP is Hz  $\times$  V/V or hertz (usually in the megahertz or gigahertz range).

In the good old days before marketers muddled the waters, you'd hear of

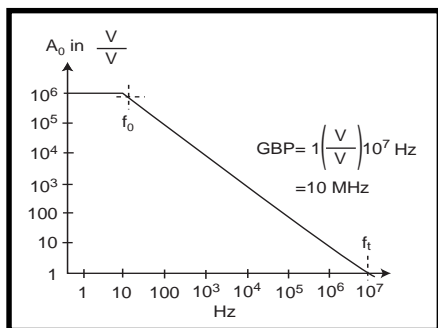


Figure 2—In internally compensated op-amps, the open-loop gain has a dominant pole at  $f_o$ .

GBP and think of the behavior shown in Figure 2. By dividing GBP by the intended closed-loop gain, you got a good idea of the bandwidth available for the application. But, not anymore.

The GBP in Figure 2 is constant anywhere along the slope of  $A_o$ . This feature is what makes GBP a useful parameter for figuring out the usable bandwidth. Not all devices have a constant GBP.

Current-feedback amplifiers (CFAs) have open-loop gain characteristics that depend on the closed-loop gain. Instrumentation amplifiers (IAs) have GBP characteristics that depend on their internal topology and the circuit gain.

GBPs for CFAs and IAs are horses of entirely different colors. When you select an op-amp for a high-frequency application, look at all device parameters, not just GBP.

For example, consider Burr-Brown's OPA643 op-amp. The datasheet for this useful, economical wide-bandwidth device boasts a GBP of 1.5 GHz, which means, qualitatively, it should be useful at relatively high frequencies.

The device is a voltage-feedback amplifier (versus a CFA). However, when you look at a graph of  $A_o$  as a function of frequency, the open-loop gain isn't as well-behaved as that in Figure 2.

The datasheet also indicates the OPA643 is a noncompensated device and only stable for closed-loop gains greater than 5 V/V. It can serve as a 10-dB video amp but is unsuitable as a unity-gain follower in a high-frequency measurement system.

In addition to the GBP, you need to consider the slew rate in high-frequency performance, which measures how fast an op-amp can swing its output voltage. The op-amp must be able to swing the output voltage as fast as the maximum derivative of the highest frequency component of the wave form. Slew rate is measured in V/ $\mu$ s.

Consider an amplifier designed to provide 10 V/V gain for input signals up to 1 Vp-p and 1 kHz. Maximum slew rate would occur on an output waveform of (1 Vp-p  $\times$  10 V/V) 10 Vp-p at 1 kHz, right at the zero crossing. So,

$$\left(\frac{d}{dt}\right)10\sin(2\pi1000t) = 10 \times 2\pi1000\cos(2\pi1000t)$$

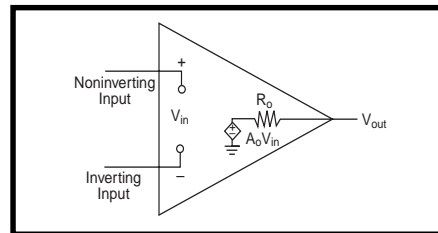


Figure 1—The first-order op-amp model is simple yet remarkably useful.

evaluated at  $t = 0$ ,  $\pi = 3$ , results in 60,000 V/s or 0.06 V/ $\mu$ s. So, your chosen op-amp for this example circuit must have a slew rate of at least 0.06 V/ $\mu$ s.

Many common op-amps (e.g., LM741, LM324, OP497) have slew rates around 0.5 V/ $\mu$ s. Any of these would suffice here. By contrast, the OPA643 has a respectable slew rate of 1000 V/ $\mu$ s.

The output resistance of the op-amp,  $R_o$ , is the final parameter I want to discuss with respect to the model shown in Figure 1.  $R_o$  is often mistaken as the output resistance of the overall amplifier circuit,  $R_{out}$ .

In typical amplifier configurations, the negative feedback desensitizes the circuit to  $R_o$ . Therefore, the output resistance  $R_{out}$  is not the same as the op-amp's  $R_o$ .

As seen in Figure 3, the feedback network attaches to the output pin of the op-amp after  $R_o$ . Negative feedback drives the output pin to the required level. The circuit's overall output resistance is:

$$R_{out} = \frac{R_o}{1 + A_o b} \quad [1]$$

The feedback factor,  $b$ , reflects the attenuation of the output voltage before it is fed back into the inverting input. For the inverting and noninverting configurations of Figure 3:

$$b = \frac{R_1}{R_1 + R_2}$$

Inverting and noninverting amplifiers have an extremely low output resistance. However,  $R_{out}$  increases as the frequency increases and  $A_o$  rolls off, which is a result of  $A_o$  in the denominator of equation 1.

$R_o$  limits the maximum current the op-amp can source or sink. Datasheets most often give a maximum value for short-circuit current. This parameter

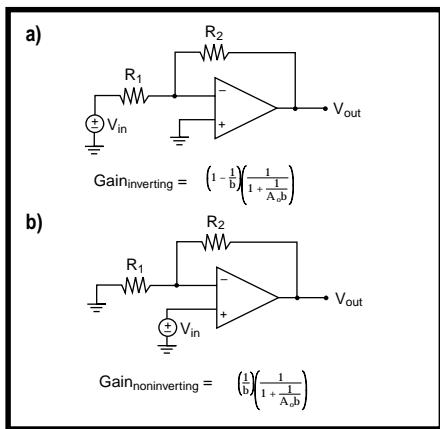


Figure 3—Op-amps can be configured as inverting (a) or noninverting (b) amplifiers.

can be used to back calculate  $R_o$  for modeling. Simply take the maximum output voltage and divide it by the maximum source current to obtain a value for  $R_o$ .

### CIRCUIT-LEVEL ANALYSIS

Figure 3 shows schematics and equations for the two most common op-amp configurations—the inverting and noninverting amplifiers. To analyze these, sum the currents at the inverting node.

To simplify the analysis, assume  $R_o$  is zero. You greatly simplify the analysis, yet still obtain useful results. Other nonideal characteristics can be analyzed separately and, through the use of superposition, can be integrated into an overall circuit model.

Figure 3 shows the equations deduced using the above technique. To further simplify, let  $A_o$  approach infinity to obtain:

$$\text{Gain}_{\text{inverting}} = -\frac{R_2}{R_1}$$

$$\text{Gain}_{\text{noninverting}} = 1 + \frac{R_2}{R_1}$$

Keep in mind that these two gain equations were derived with much simplification. They don't reflect the effects of the open-loop gain rolloff. However, these are the most commonly used equations for describing the behavior of the circuits in Figure 3.

When the frequency response of  $A_o$  is relevant

(e.g., when high closed-loop gains are needed), use the equations in Figure 3. I recommend simulation tools like SPICE or MicroCap when you require highly accurate predictions.

SPICE models take into account second-order effects like bias currents, offset currents, and offset voltage. You can find detailed op-amp models for virtually every commercially available op-amp. Most manufacturers supply the models free of charge on the Internet or on CD.

A common concept used to analyze op-amp circuits is the idea of a “virtual ground” or “virtual short,” which mean the same thing. The voltage on the op-amp's inverting node is forced by the negative feedback to be at the same potential as the noninverting node.

This concept assumes that  $A_o$  approaches infinity, therefore requiring the difference between the inverting and noninverting inputs to approach zero for  $V_{out}$  to be finite.

As with all approximations, there are times when this one is useful and times when it's not. Figure 2 shows  $A_o$  is generally pretty large at low frequencies. So, the virtual-short idea is most useful at low frequencies.

In the case of the inverting amplifier, the term “virtual ground” is applied to the inverting input of the op-amp. The noninverting node is held at ground. Negative feedback forces the inverting node to be at the same potential as the noninverting node (i.e., ground). And, the inverting input of the amplifier is said to be a virtual ground.

The phrase “virtual short” is used for the noninverting amplifier. The input signal is applied at the noninverting node, and the inverting input

is forced to this same potential. Since the two nodes are held at the same potential by the negative feedback, and that potential is not necessarily zero, the inputs of the op-amps are said to have a virtual short across them.

### SECOND-ORDER EFFECTS

Now that we have examined op-amps at the circuit level, let's go on to modeling and predicting behavior due to second-order effects. Developing an intuitive feel for these effects enables you to select devices best suited to your application.

Figure 4a shows how to model op-amp input bias and offset currents.  $I_{\text{bias}}$  is the tiny base current (or gate leakage) the internal transistors need for operation. This value is orders of magnitude smaller for FET-based input stages than for BJT input stages [1].

$I_{\text{offset}}$  results from the mismatch between the input transistors.  $I_{\text{offset}}$  is typically one order of magnitude smaller than  $I_{\text{bias}}$  [1].

The polarity of  $I_{\text{bias}}$  is constant and predictable if the topology of the input stage is known. However,  $I_{\text{offset}}$  may be of either polarity.

To examine the effect of these currents, remove the stimulus from the circuit and look at the  $V_o$  generated by the offset currents. With the inputs grounded, the inverting and noninverting configurations become the same circuit. Figure 4b gives you a glimpse of the circuit, assuming that  $R_{\text{compensation}} = 0$ .

This configuration offers no hope for nulling offsets introduced by bias currents, which is why  $R_{\text{compensation}}$  has been added. Analyzing the circuit with  $R_{\text{compensation}} \neq 0$  yields:

$$V_o = \left(1 + \frac{R_2}{R_1}\right) \times \left[ (R_1 \parallel R_2) I_- - R_{\text{compensation}} \times I_+ \right]$$

where  $I_+ = I_{\text{bias}} + I_{\text{offset}}$  and  $I_- = I_{\text{bias}} - I_{\text{offset}}$ .

To compensate for the bias currents, simply set  $R_{\text{compensation}}$  equal to the equivalent resistance of  $R_1$  in parallel with  $R_2$ . However, this

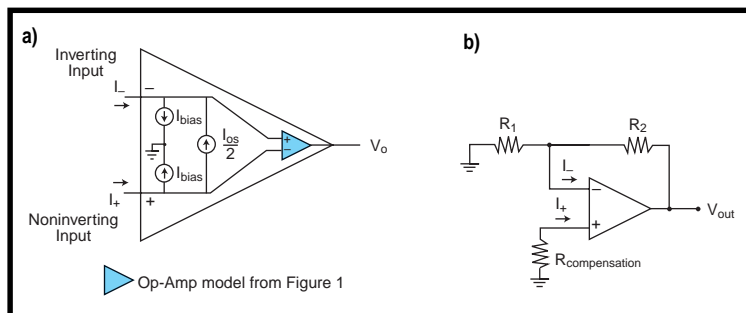
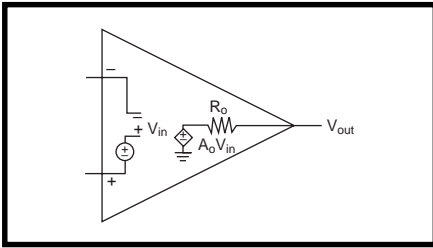


Figure 4a—Input bias and input offset currents are modeled as separate independent sources. b— $R_{\text{compensation}}$  provides a means of nulling the offset in  $V_{out}$  caused by  $I_{\text{bias}}$ .



**Figure 5**—Input voltage offset is modeled by the inclusion of an independent voltage source in the simple model shown in Figure 1.

doesn't compensate for  $I_{offset}$ . The overall error reduces to:

$$V_o = \left(1 + \frac{R_2}{R_1}\right) \left[-2(R_1 \parallel R_2)I_{offset}\right]$$

Since  $I_{offset}$  is typically an order of magnitude less than  $I_{bias}$ , adding  $R_{compensation}$  (equal to  $R_1 \parallel R_2$ ) is usually sufficient.

Other methods to reduce error due to  $I_{offset}$  are to:

- keep  $R_1 \parallel R_2$  small
- keep  $R_2$  small

Both increase power dissipation. The tradeoff may seem simple, but a quick look at a graph of  $I_{bias}$  versus dice temperature reveals the insidious truth. Many op-amps, especially those with FET input stages, increase  $I_{bias}$  exponentially as a function of temperature.

Devices with superbeta input stages, like the OP297 and OP497 from Analog Devices, offer a good compromise between  $I_{bias}$  and the temperature coefficient. Superbeta input stages have moderately low initial  $I_{bias}$  currents and do not suffer from an exponential temperature dependence.

Increased power means increased dice temperature. Evaluate the tradeoffs carefully.

$I_{offset}$  is related to input current noise. Intrinsic noise on the dice creates a small amount of fluctuation in  $I_{offset}$ . This current noise will develop into a voltage across  $R_1 \parallel R_2$  and be multiplied by:

$$A_{noise} = \left(1 + \frac{R_2}{R_1}\right)$$

The noise gain ( $A_{noise}$ ) is the same for both inverting and noninverting configurations.

The op-amp input current noise is typically very tiny—subpicoamps.

However, the voltage noise at the output can become significant as current noise is developed into a voltage across  $2 \times R_1 \parallel R_2$  and multiplied by  $A_{noise}$ .

Op-amp noise is frequency dependent. Datasheets always list tabulated data, and better datasheets have graphs.

Another source of error in op-amps is the offset voltage ( $V_{os}$ ). The cause of  $V_{os}$  is the unavoidable mismatch of devices and operating points on the dice. Figure 5 shows the model for offset voltage.

$V_{os}$  is highly temperature dependent. The polarity is unpredictable and may reverse over temperature.

Using the model in Figure 5 with the circuit topology from Figure 4b (with  $R_{compensation} = 0$ ), I derive:

$$V_o = \left(1 + \frac{R_2}{R_1}\right) V_{os} = A_{noise} \times V_{os} \quad [2]$$

Compensating for  $V_{os}$  can be accomplished via external nulling (i.e., introducing an external voltage at one of the op-amp inputs) or internal nulling (i.e., unbalancing one of the internal differential pairs).

External nulling can be accomplished by using a trimmer, digitally controlled pot, or DAC to dial in the required compensation voltage. Internal nulling is accomplished by connecting a

trimmer to the offset pin or pins on the op-amp. The device datasheet has trimmer-value recommendations and suggested configurations.

A common problem occurs when multiple op-amps are chained in a circuit. Each op-amp contributes an offset to the system. The most common question asked is, "Do I null each op-amp's  $V_{os}$  or null the overall system at a single point?"

I lean toward nulling each op-amp using internal nulling. I definitely shy away from nulling the entire system offset using the internal nulling on a single stage. The latter is much simpler but unreliable over temperature.

If you use internal nulling to introduce a huge imbalance in a single stage to compensate for the total system offset, you are assured of temperature-tracking problems. The grossly unbalanced op-amp can't track the overall system offset.

If you null each stage internally, the system offset remains as stable as practically possible over temperature.

With the DC offset nulled, let's turn our attention to noise on  $V_{os}$ . All  $V_{os}$  noise, regardless of origin, shows up in the amplifier output multiplied by  $A_{noise}$  (see equation 2).

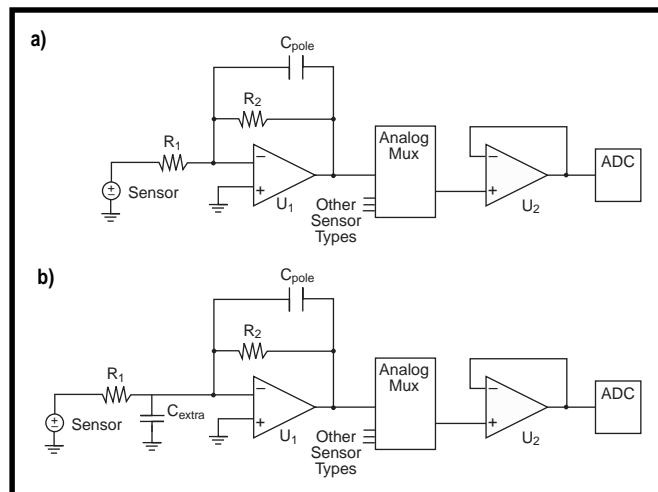
Intrinsic noise on the dice causes perturbations in  $V_{os}$ , which is referred to as input voltage noise ( $E_{os}$ ). Datasheets give tabulated data for  $E_{os}$ .

The power supply induces another  $V_{os}$  noise. As the power-supply voltage fluctuates, the internal bias points of the op-amp shift. Because  $V_{os}$  is the result of internal bias-point mismatches, power-supply-induced noise is modeled as noise on  $V_{os}$ .

The power-supply rejection ratio (PSRR) is:

$$PSRR = \left(\frac{\Delta V_{os}}{\Delta V_{supply}}\right)$$

Another culprit introducing offset error is the common-mode voltage on the op-amp's inputs. As the voltage on the inputs is raised or lowered, the op-amp's internal bias points shift and additional  $V_{os}$



**Figure 6a**— $R_2$  and  $C_{pole}$  form a single-pole filter. **b**—A virtual ground exists at the inverting input of  $U_1$ , so  $C_{extra}$  has a virtual short across it.

develops. Thus, common-mode noise can be modeled as noise on  $V_{os}$ .

The common-mode rejection ratio (CMRR) is:

$$CMRR = \left( \frac{\Delta V_{os}}{\Delta V_{\text{common-mode}}} \right)$$

Both PSRR and CMRR generally roll off with frequency. Some devices—especially chopper-stabilized amps—have a notch in the rolloff. Always refer to graphs of these parameters, and don't forget that CMRR and PSRR are referenced to  $V_{os}$ .

It's incorrect to reason that if you have  $x V_{RMS}$  on your rail and the PSRR is -60 dB, your output will only see  $x/1000$  V of noise. Wrong!

Instead, you need to think, "If I have  $x V_{RMS}$  of noise on my rail, my noise gain is 40 dB, and my PSRR is -60 dB, then my output will see  $x/10$  V of noise."

Well, the basics review is over. Now, let's look at some real-life examples.

## WATCH OUT FOR VIRTUAL SHORTS

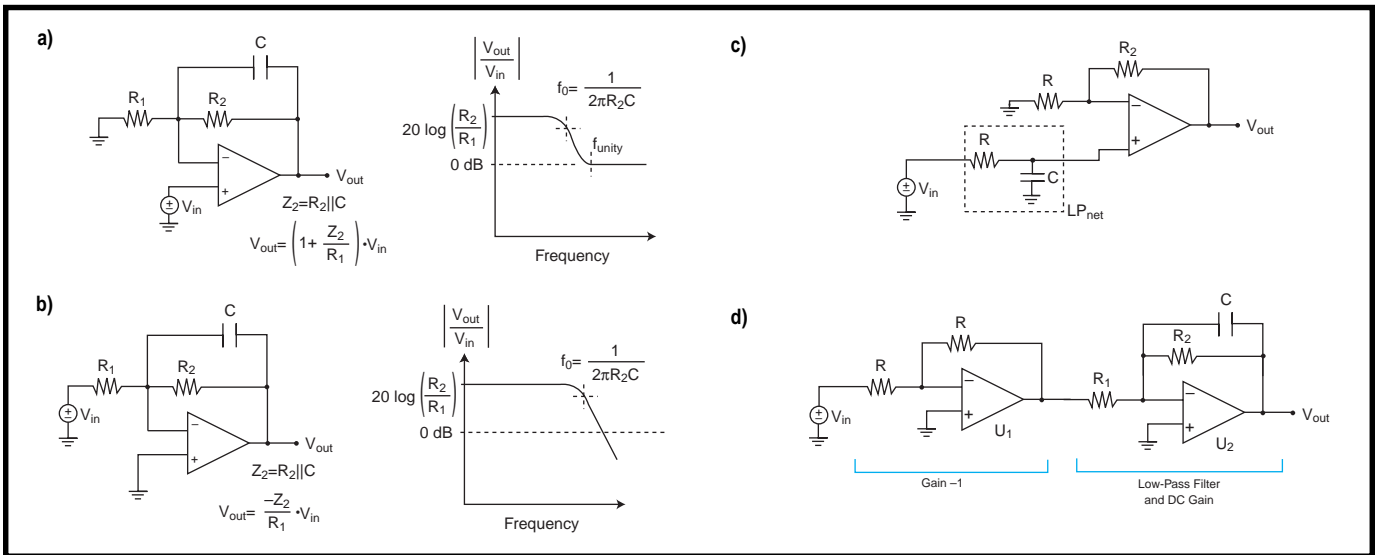
A number of years ago, I was developing a hand-held instrument with optical sensors on a meter-long probe. The signal paths to the sensor formed long loop antennae. The functional geometry for the instrument required this suboptimal electrical configuration.

In my signal path, I had the single-pole low-pass filter/inverting amplifier shown in Figure 6a. I needed an additional pole to provide sufficient band limiting of noise.

I added a capacitor as shown in Figure 6b. I expected  $R_2$  and  $C_{\text{pole}}$  to form the first pole, and  $R_1$  and  $C_{\text{extra}}$  to form the second pole. On testing the circuit, I found it behaved as only a single-pole ( $R_2$  and  $C_{\text{pole}}$ ) circuit.

After much wailing and gnashing of teeth (and a little algebra), I realized that  $C_{\text{extra}}$  is attached between a virtual ground and ground.  $C_{\text{extra}}$  had a virtual short across it and was therefore superfluous. I added my second pole with a passive RC on the output and obtained satisfactory results.

As it turns out,  $C_{\text{extra}}$  isn't entirely superfluous. The virtual-ground concept is predicated on the assumption that



**Figure 7a**—The noninverting configuration has an intrinsic zero and does not allow attenuation below 0 dB. **b**—The inverting configuration permits attenuation below 0 dB. **c**—LPnet continues to attenuate noise from  $V_{in}$  independently of the op-amp gain stage. **d**—A two-stage antialias filter can provide high gain, low noise, low  $R_{out}$ , and excellent frequency rolloff.

$A_o \gg A_{closed\ loop}$ . As  $A_o$  rolls off and the virtual ground degrades,  $C_{extra}$  enters into the equation. The best way to analyze these kinds of second-order effects is with SPICE.

## ANTI\_ALIASING\_FILTER\_TRADEOFFS

The noninverting amplifier has an interesting feature—an intrinsic zero. By virtue of the circuit topology, there is a zero that prohibits the closed-loop gain from dropping below 1 V/V (0 dB).

The easiest way to see this is to review at the transfer function for a noninverting amplifier:

$$\frac{V_{out}}{V_{in}} = 1 + \frac{Z_2}{Z_1}$$

Regardless of what  $Z_2$  and  $Z_1$  do (short of going negative), the gain is always larger than unity. Keeping this in mind, let's compare the inverting and noninverting topologies for use as antialiasing filters.

Figure 7 shows four circuits used as antialiasing filters. Figure 7a allows all high-frequency noise beyond  $f_{unity}$  to be aliased back into the pass band without attenuation. The circuit in Figure 7b has inverted gain but is vastly superior to the circuit in Figure 7a at attenuating high-frequency noise.

The circuit in Figure 7c uses the op-amp only as a gain block. A passive network (LPnet) provides the antialias function. This configuration doesn't

attenuate the noise introduced into the system by the op-amp, but other noise is band limited by LPnet. The circuit in Figure 7b is again superior to Figure 7c because  $V_{os}$  noise is attenuated.

The weakness in the circuit in Figure 7c is the LPnet preceding the gain block. You might argue that LPnet could follow the gain block for improved noise attenuation.

But this newly proposed configuration doesn't present a low impedance to the ADC. The ADC may load LPnet, thus introducing more error.

For example, the ADC's sample and hold (S/H) may gulp current out of LPnet, and enter hold mode before LPnet can deliver sufficient charge to bring the voltage sampled up to the proper value. A good rule of thumb is to have an op-amp driving the input of the ADC's S/H.

If the  $V_{os}$  noise (intrinsic, power-supply induced, and common mode) in the circuit shown in Figure 7c multiplied by  $A_{noise}$  is suitably small for the application, then this circuit is an acceptable solution.

If signal inversion is important and high gain is required, you can precede the circuit of Figure 7b with a unity-gain inverting stage (see Figure 7d). The new stage introduces a small amount of noise.  $A_{noise}$  for a unity-gain inverting stage is only 2 V/V.

The filter portion (U2 in Figure 7d) band-limits the newly introduced noise. As well, another pole can be added to the filter by placing a capacitor in the feedback loop of the unity-gain inverting stage.

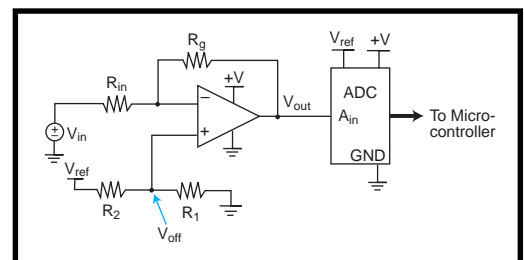
When you're designing filters around a noninverting amplifier, always keep in mind the zero that exists as an artifact of the topology.

## SINGLE-ENDED\_ANALOG\_STAGE

Figure 8 illustrates a versatile single-ended analog front end. The op-amp and ADC can be powered by a single rail supply while accepting bipolar or unipolar inputs. Tiny signals can be expanded to fill 0– $V_{ref}$ . Large input signals can be compressed. The input signal need not be center around zero.

For example a 4–20-mV range can be mapped into 0 –  $V_{ref}$ . If you select the resistor values and  $V_{ref}$  properly, this circuit serves almost any application.

Input resistance versus gain is the major tradeoff to consider with this



**Figure 8**—This versatile interface maps an arbitrary input range into 0– $V_{ref}$ .

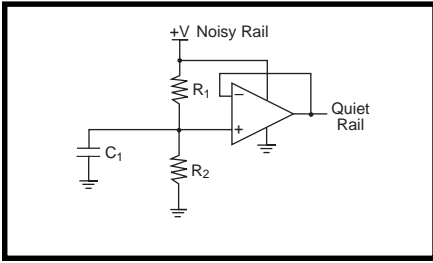


Figure 9—A spare op-amp can be used as a low-cost voltage regulator.

circuit.  $R_{in}$  must be large enough so the circuit doesn't load the sensor, but small enough to allow the ratio  $R_g/R_{in}$  to supply sufficient gain.

Designing with this circuit is simple:

- determine the sensor's output voltage range
- determine the ADC's input range
- determine the required gain (or attenuation). Deduce ideal values for  $R_g$  and  $R_{in}$ .
- determine required  $V_{off}$ . Deduce ideal values for  $R_1$  and  $R_2$ .
- select standard resistor values
- verify input range maps to output range with the selected resistor values
- repeat the last two steps until satisfied

Let's walk through the procedure. The governing equation is:

$$V_{out} = (V_{off} - V_{in}) \frac{R_g}{R_{in} + V_{off}} \quad [3]$$

The first step is to figure out what the application gain needs to be:

$$|g| = \frac{V_{ref}}{V_{inmax} - V_{inmin}}$$

From equation 3, you see that if  $V_{off}$  is 0, the circuit gain is:

$$g = -\frac{R_g}{R_{in}} \quad [4]$$

Now, select  $R_g$  and  $R_{in}$ .  $R_{in}$  should be as high as possible to avoid loading the sensor.

Next, determine  $V_{off}$ . Note that the circuit is an inverting amplifier. When

$V_{inmax}$  is applied, the desired  $V_{out}$  is zero. Thus from equation 3, you obtain:

$$0 = (V_{off} - V_{inmax}) \frac{R_g}{R_{in} + V_{off}} \quad [5]$$

Using equation 4 and rearranging equation 5, you can determine  $V_{off}$ :

$$V_{off} = V_{inmax} \left( \frac{|g|}{1 + |g|} \right)$$

Now, pick  $R_1$  and  $R_2$ .  $R_1$  and  $R_2$  form a voltage divider that dials in  $V_{off}$ . You can select  $R_1$  and  $R_2$  by using:

$$V_{off} = \frac{R_1}{R_1 + R_2} \times V_{ref}$$

Finally, pick standard resistor values and use equation 3 to verify that the design maps the  $V_{sensor}$  range in to the  $V_{ADC}$  range.

If a pole is needed, you can add one by placing a capacitor in the feedback loop.

The circuit in Figure 8 is shown as an input interface. However, the same topology can be used to condition the output of a DAC. A bipolar output is possible if a negative rail is available.

You can use this circuit to map any arbitrary input range to any arbitrary output range. This is restricted only by practical limits of available components.

### COST-EFFECTIVE LOW-NOISE RAIL

To supply clean power to low-noise analog circuits, you can add a linear regulator just for the analog section. Modern micropower amplifiers don't need huge amounts of current. Regulators are bulky, and dropout voltage is always a concern. In many cases, a dedicated regulator is overkill.

In some cases, the nifty circuit in Figure 9 may be an almost free solution. It uses the following concepts to provide a low-noise power rail [2]:

- PSRR characteristics of the op-amp
- $R_o$  desensitivity
- the high input impedance of the noninverting node
- low  $A_{noise}$

The op-amps are capable of sourcing several milliamperes, which is often enough current to drive many micro-power analog stages. An op-amp in a SOT-23/5 package and a couple 0402 passives may occupy less board space than a bulky linear regulator.

If the circuit already has an extra op-amp (e.g., in an existing dual or quad package) powered from a noisy rail, the two resistors and capacitor are all you need for a low-noise power rail for your quiet analog section.

### INSTRUMENTATION AMPLIFIERS

Instrumentation amplifiers (IAs) are differential. They have a schematic symbol similar to op-amps, but are not op-amps. Figure 10 shows its symbol and a three op-amp implementation of it. IAs can be built from discrete components or purchased as single chips.

The voltage between the inverting and noninverting inputs is amplified by the IA's gain. This gain is usually set by a single resistor,  $R_g$ . Single-chip IAs are often capable of 1–1000 V/V gains.

The IA's two input op-amps are configured as noninverting amplifiers. They provide the IA's high-impedance input characteristics that it's known for. The output op-amp is configured as a differential amp. The overall IA gain is:

$$IA_{gain} = \left[ 1 + 2 \left( \frac{R_1}{R_g} \right) \right] \left( \frac{R_3}{R_2} \right)$$

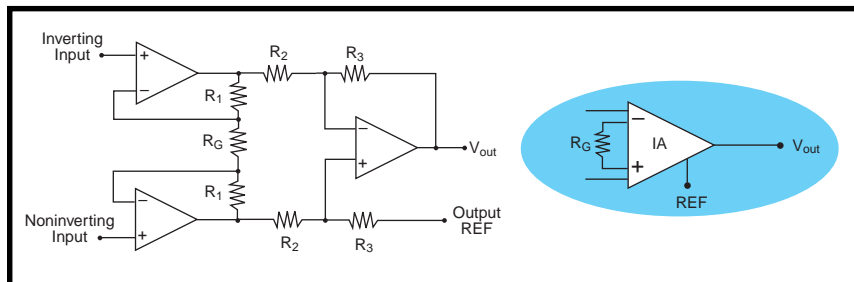


Figure 10—Today, \$3 can buy a single monolithic chip with a full-blown instrumentation amplifier.

Although other implementations of IAs exist [1], the one in Figure 10 is the most widely known.

The advent of the single-chip monolithic IA is a boon to instrumen-

tation engineers. In the not-so-good old days, a circuit designer had to build an instrumentation amp from discrete op-amps and resistors. It looked pretty good on paper, but in practice, matching the discrete resistors was difficult and performance suffered.

For an IA to achieve CMRRs on the order of 120 dB, the resistors must be precisely matched in value and temperature coefficient. The mismatches associated with 1% resistors in discrete designs were often unacceptable in high-precision applications. Also, careful consideration was needed to keep the resistor pairs isothermal.

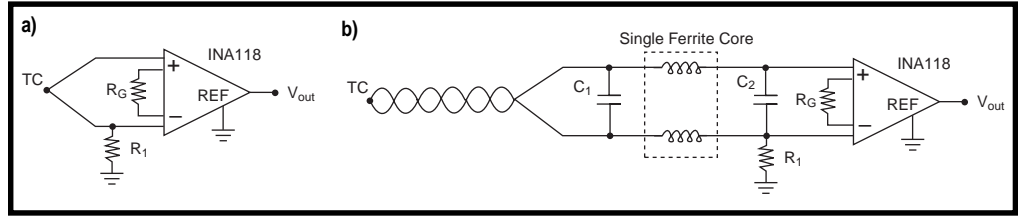
The single-chip IAs offer superbly matched components. With all the parts on a single die, everything stays isothermal. Laser trimming of internal resistors guarantees maximum precision. Monolithic IAs save board space, reduce component count, save money, and maximize performance.

Some manufacturers have single-chip IAs with chopper-stabilized front ends. These devices are designed to minimize  $V_{os}$ . For example, the LTC1100 from Linear Technologies has an admirable  $V_{os}$  of 10  $\mu$ V. This device has an internal oscillator and internal caps to support the chopper stabilization.

Figure 10 shows that both inputs of the IA have a high impedance. Notably, 10 G $\Omega$  is a fairly common equivalent input resistance for an IA. This, coupled with high CMRR and a wide range of gains, makes the IA an easy-to-use and versatile tool.

### IA THERMOCOUPLE CONDITIONER

A thermocouple is a junction of dissimilar metals that produces a voltage proportional to temperature. Thermocouples typically have an absolute accuracy of only a degree or two Celsius. However, thermocouple mea-



**Figure 11a**—A low-cost instrumentation amplifier can provide a nearly ideal thermocouple interface. **b**—A common-mode filter and twisted-pair sensor cable clean up induced common-mode noise.

surements are extraordinarily stable and repeatable.

In a calibrated system, temperature  $\Delta$ s of 0.001 $^{\circ}$ C can be reliably measured with thermocouples. Figure 11a shows a simple thermocouple (TC) interface. R1 prevents the high-impedance inputs from developing sufficient charge to drift outside of the supply rails.

Figure 11b offers a remedy for noise induced on the sensor leads. The ferrite bead and two capacitors form a common-mode filter. The twisted pair ensures noise is picked up equally on both conductors.

Briefly, let's explore how much loading error is introduced into the measurement by the IA and common-mode filter in Figure 11b. The leakage resistance of an X7R dielectric ceramic capacitor is about 10 G $\Omega$ . The addition of the two capacitors cuts the DC input resistance seen by the thermocouple by two-thirds, putting about a 3-G $\Omega$  load on the thermocouple.

Thermocouples are often modeled as a voltage source with a series source resistance,  $R_s$ . The size and type of junction determine the value of  $R_s$ . For my needs, 10  $\Omega$  is a reasonable value for  $R_s$ .

The load resistance (3 G $\Omega$ ) and  $R_s$  (10  $\Omega$ ) form a voltage divider, with the tap of the divider being the actual voltage observed. The loading error is only about three parts in a billion. This is a negligible effect for thermocouple applications.

Minimal loading of the TC, high CMRR, high PSRR, and differential front end are needed to measure tem-

perature changes on the order of millidegrees Celsius. IAs provide a simple and economical solution for applications measuring small temperature changes.

For example, consider a microcalorim-

eter. The maximum expected  $\Delta$  is 10 $^{\circ}$ C. The typical output of a type-E thermocouple is 60  $\mu$ V/ $^{\circ}$ C. If the system has a 2.5-V full-scale ADC, then the gain on the IA must be:

$$\frac{2.5}{10 \times 60 \times 10^{-6}} = 4166 \text{ V/V}$$

This is achievable with a single \$3 eight-pin IA from Burr Brown—the INA118.

### CHOP TO NULL SYSTEM OFFSET

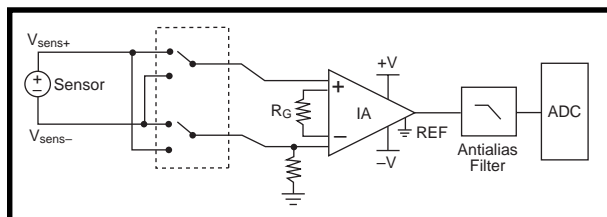
Offset voltage is still a caveat. IAs have an offset voltage much like op-amps. For many IAs, this offset is around 100  $\mu$ V. In high-gain systems like a microcalorimeter,  $V_{os}$  can be a major problem (4166  $\times$  100  $\mu$ V = 0.42 V of offset at the output).

Chopping the input is a technique for reducing offset voltage. Two measurements are made of different quantities. Since the offset voltage shows up in both measurements, a subtraction drops out the offset voltage.

Many chopper-stabilized ICs make a measurement of the input voltage, then internally measure a ground potential. The two measurements are subtracted and presented to the output. All of this goes on internally and is transparent to the external circuit.

For high-gain applications, selecting an IA with low  $V_{os}$  is a good start. Two low-offset nonchopper-stabilized IAs are Analog Devices' AD620B and Burr Brown's INA118 with a  $V_{os}$  of 50  $\mu$ V. The INA118's  $V_{os}$  depends on the gain setting and can be as much as 300  $\mu$ V at unity gain. The Linear Technology LTC1100 is an internally chopper-stabilized IA with a  $V_{os}$  of only 10  $\mu$ V.

Chopping externally to the IA is an outstanding and time-proven technique for reducing channel offset. Figure 12 shows a signal-conditioning circuit that uses chopping to null the system offset.



**Figure 12**—Chopping the input signal enables the nulling of system offset in software.

This technique nulls the offset associated with the entire measurement channel, not just the IA  $V_{os}$ .

The sequence of events in a measurement cycle is:

- the input mux is set such that  $V_{sens+}$  routes to the noninverting input and  $V_{sens-}$  routes to the inverting input
- a measurement is taken with the ADC:  $M1 = (V_{sensor} + V_{os}) \text{ Gain}$
- the input mux is reversed such that  $V_{sens+}$  routes to the inverting input and  $V_{sens-}$  routes to the noninverting input.
- a measurement is taken with the ADC:  $M2 = (-V_{sensor} + V_{os}) \text{ Gain}$
- compute:

$$V_{sensor} = \frac{M1 - M2}{2 \times \text{Gain}}$$

When  $M_1 - M_2$  is computed, the  $V_{os}$  introduced into the measurement drops out.

For chopping to work, measurements  $M_1$  and  $M_2$  should be taken as closely together in time as possible. Normally, the limiting factor is the settling time of the system's antialiasing filter. Burr-Brown has an excellent app note on fast-settling low-pass filters [3].

A major advantage of the chopped input technique is the ability to null the system offset with each measurement. In real-world systems, system offset voltages drift with time and temperature. This technique enables the system to compensate for the unpredictable drift of offsets.

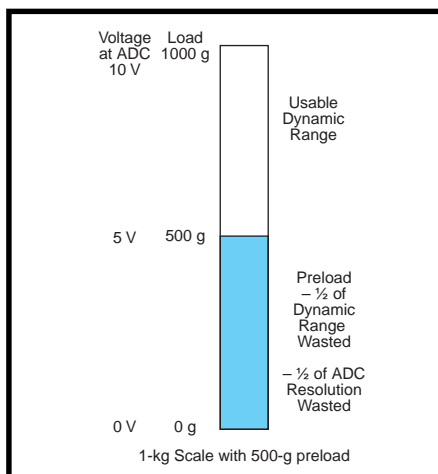


Figure 14—ADC resolution and dynamic range are expensive. Don't trade them away in a simplistic nulling scheme.

## IA RESISTIVE BRIDGE SENSOR INTERFACE

High input impedance makes the IA ideal for measuring resistive bridge sensors. Strain gages, pressure transducers, RTDs, and load cells are common examples of resistive bridge sensors.

Bridge transducers are balanced or unbalanced. Balanced bridges use feedback to force the voltage on the differential nodes to be the same by tweaking one or more resistive elements in the bridge. Unbalanced bridge systems simply measure the voltage on the differential nodes.

Both systems rely on precise measurement of the voltage across the differential nodes. This work demands a high-impedance differential amplifier—an IA.

Figure 13 shows a measurement system using an IA as the bridge interface. The IA amplifies the voltage across the differential nodes of the bridge. The ADC measures both the output of the IA and the excitation voltage on the top of the bridge.

From these two quantities, the controller determines the magnitude of the physical stimulus on the bridge. The exact determination of the value depends on the bridge characteristics.

Typical strain gages have bridge elements between 120 and 350  $\Omega$  [4]. The IA has an input impedance a hundred million times greater, and the loading effect is negligible.

In a strain-gage or RTD application, the full-scale differential output voltage may only be 10 mV. IA gains of 60 dB (1000 V/V) are therefore required to bring the signal up to typical working levels, but that's not a problem for IAs like the AD620.

As with high-gain thermocouple applications,  $V_{os}$  must be considered in the system design. For example, the 50- $\mu\text{V}$   $V_{os}$  of the AD620 times 60 dB yields an output-voltage offset of 50 mV.

If the full-scale output voltage is 10 V and measured with a 12-bit ADC, the offset introduced by the IA is 20.4 codes out of 4096. This error is manageable and can be removed in software.

Bridge transducers are often located on long cables, which tend to pick up common-mode noise. Multiples of 50 and 60 Hz are the most common.

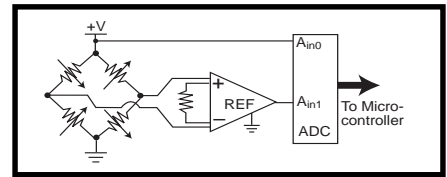


Figure 13—Instrumentation amplifiers can measure unbalanced bridge transducers without significant loading.

The high CMRR of IAs effectively reduces this unwanted noise. Common-mode RF can be significantly attenuated by ferrite beads (see Figure 11b).

The single-chip IA greatly simplifies the job of the analog instrumentation engineer. Measuring bridge circuits is still considered somewhat of an art. For most of us, precanned IAs reduce the problem to the paint-by-numbers level.

## PRELOAD COMPENSATION

A sensor preload is a load that is undesired but necessary for the measurement. For example, a load cell in a digital scale may use a bowl to hold the product being measured. The bowl is a preload.

One way to null a preload is to sample the sensor with the preload and subtract the preload from subsequent measurements in software. At first glance, this software nulling technique seems like a good solution. But, the tradeoff for simplicity is dynamic range.

For example, consider a system with a 0–10-V ADC input corresponding to a 0–1000-g stimulus on the load cell. If a 500-g bowl is placed on the load cell, the ADC sees a 5-V preload. The remaining 5–10-V range is still available for measurement, but half the system's dynamic range is lost to the preload (see Figure 14).

If a 1000-g limit is determined by the system gain settings and not the load cell's capacity, you can restore the dynamic range of the system via a preload-compensation circuit. For Figure 15's circuit to work, the load cell must measure the maximum expected preload plus the maximum load due to product.

For the sake of discussion, let's say the system needs to measure 0–1000 g of product with up to a 1000-g preload. So, the load cell must be capable of measuring 2000 g.

The microcontroller sets the DAC and mux to generate a compensating



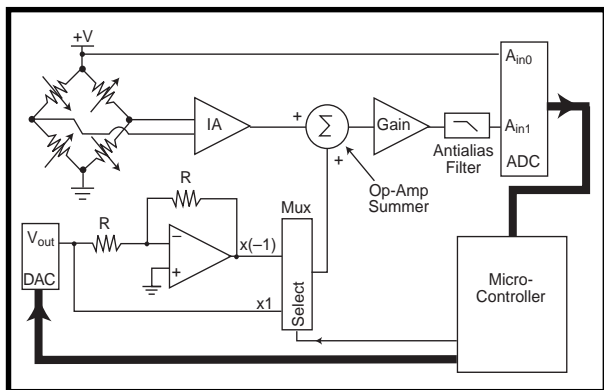


Figure 15—Preload compensation can be accomplished without sacrificing dynamic range.

voltage equal to the preload voltage on the sensor (but of opposite polarity). The summer adds the sensor output and compensating voltage. Once the DAC section is set up, the output of the summer is zero when the preload is present.

The ADC now sees 0 V when the bowl is preloading the load cell. As product is placed in the bowl, the load on the sensor increases and the summer's output increases. The ADC sees a voltage corresponding only to the product weight, and the full 0–10-V range is available for measuring product. The system's usable dynamic range is restored.

This technique differs from gain-switching schemes because the product placed on the sensor is always measured in a zero to full-scale range for the ADC. So, regardless of the preload, the scale always has the same resolution. Using the full range of the ADC for product measurement ensures maximum resolution.

There are limitations to this technique based on sensor linearity and DAC versus ADC resolution. Although I've simplified the technique, you can get a good start with the topology shown in Figure 15.

Gain-switching schemes can be combined with this nulling technique to provide an overall system capable of incredibly fine measurement resolution while accommodating a huge preload.

If the Gain block in Figure 15 is programmable, we could have a high-gain mode and a low-gain mode. For example, low-gain mode, may map 0–1000 g of sensor stimulus to 0 to full-scale on the ADC. High-gain mode

may map 0 to 1g of sensor stimulus to 0 to full-scale on the ADC.

Consider again the case of the 500-g bowl on the sensor. To compensate for the preload:

- the microcalorimeter selects low-gain mode
- the 500-g bowl is measured, and the DAC section is set to null the preload
- the microcalorimeter selects high-gain mode
- the microcalorimeter tweaks with the DAC until any remaining preload disappears (this step assumes the DAC is very high resolution, which is a simplification)

Next the user places product—e.g., 25 mg of salt—into the bowl. In high gain mode, the system gives the user maximum resolution in the 0–1-g range.

If the user pours another 750 g of salt into the bowl, the system switches to low-gain mode and gives a resolution corresponding to the 0–1000-g range.

The technique in Figure 15 is a versatile and powerful method of preload compensation. For applications where 100% of the ADC's dynamic range must be used, Figure 15 offers a good starting point. For less-demanding applications, a combination of gain switching and pure software nulling suffices.

## INFO YOU CAN USE

I've reviewed op-amps, as well as examining some interesting mistakes and some excellent interface circuits. I didn't discuss PCB-layout considerations, but this topic is well-covered elsewhere [5].

As you've seen, analog design is full of tradeoffs. Models help, and SPICE is a great tool. But, nothing beats gaining insight and intuition by examining other people's successes and failures. ☐

*Bob Perrin currently works for Z-World. He has designed mixed signal instrumentation for agronomy research, food science, harsh-environment industrial computing, and embedded con-*

*trol. You may reach Bob at bperrin@zworld.com.*

## REFERENCES

- [1] S. Franco, "Practical Op-Amp Limitations," *Design with Operational Amplifiers and Analog Integrated Circuits*, McGraw-Hill, New York, NY, 188–241, 1988.
- [2] C.D. Motchenbacher and J.A. Connelly, "Capacity Multiplier Filter," *Low Noise Electronic System Design*, John Wiley & Sons, New York, NY, 309–310, 1993.
- [3] Burr-Brown, *Fast Settling Low Pass Filter*, App Note AB-022, 1994.
- [4] K. Hoffmann, "Selection Criteria For Strain Gages," *An Introduction to Measurements Using Strain Gages*, Hottinger Baldwin Messtechnik GmbH, Darmstadt, Germany, p. 58, 1989.
- [5] B. Baker and J. Graeme, "Systematic Approach Makes Op-Amp Circuits Resist Radiated Noise," *EDN*, **40:15**, p. 93, 1995.

## SOURCES

### OP297 and OP497 op-amps, AD620B instrumentation amplifier

Analog Devices  
One Technology Way  
Norwood, MA 02062-9106  
(781) 329-4700  
Fax: (781) 461-4261  
www.analog.com

### OPA643 op-amp, INA118 instrumentation amplifier

Burr-Brown Corp.  
P.O. Box 11400  
Tucson, AZ 85734-1400  
(520) 746-1111  
Fax: (520) 741-3895  
www.burr-brown.com

### LTC1100 instrumentation amplifier

Linear Technology Corp.  
1630 McCarthy Blvd.,  
Milpitas, CA 95035-7487  
(408) 432-1900  
Fax: (408) 434-0507  
www.linear-tech.com

## I R S

- 401 Very Useful
- 402 Moderately Useful
- 403 Not Useful

# FEATURE ARTICLE

Mike Zerkus

## Antialiasing Techniques

To get rid of aliasing in an ADC system, you can use linear active filters, switched-capacitor filters, .... You also better know when a filter isn't the answer. So, check out Mike's hints for anticipating as well as solving aliasing problems.



Aliasing can contaminate any analog-to-digital process, but it's poorly understood by most engineers. Much of the time, aliasing causes poor data quality. In the worst case, aliased data is accepted and analyzed as fact without anyone ever suspecting that the data is bogus.

But, there's good news, too. Aliasing can be spotted and eliminated if you apply a few simple rules of thumb.

Aliasing is the creation of a false, low-frequency signal. The false signal is the result of an insufficient sample rate of a high-frequency signal.

Aliasing can take many forms. One common example is the apparent backward motion of spoked wagon wheels in old western movies. The wheels appear to spin backward because the camera's frame rate is slower than the wheels' angular velocity.

As the wagon slows down, the wheels apparently change direction, speed up, slow down, and stand still until the wagon is slow enough that the camera can capture the true motion of the wheels. Strictly speaking, this is called temporal aliasing [1].

Spatial aliasing is the jagged edge you sometimes see on lines drawn on a computer screen. A chord played on

a musical instrument is another type of aliasing, pleasing to the ear if done correctly.

Of interest here, however, is aliasing that affects A/D conversion. In this article, I assume you have a prior knowledge of ADCs, a general knowledge of filters, and a desire to cut through the theory and get to a few helpful tips. However, if you need a refresher, check out Bob Perrin's "High-Resolution ADCs" (*INK* 74) or *Design of Active Filters with Experiments* [2].

### ALIASING IN AN ADC SYSTEM

Aliasing most often appears as a very low-frequency roll in the data, almost indistinguishable from a DC drift. The aliasing is caused by noise whose frequency is greater than the Nyquist frequency.

The noise is undersampled, and the difference between the noise and the Nyquist frequency appears in the data. The amplitude of the false signal depends on the amplitude of the noise.

### THE NYQUIST FABLE

The Nyquist rule says that the minimum sample rate required to describe a signal is at least two times the frequency of interest [3]. This rule only works when the signal to be sampled has no frequency component greater than half the sample rate [4].

In the real world, everything has noise. So, every frequency above half the sample rate is aliased or folded back and appears as low-frequency components of the signal. The closer in frequency the noise is to the Nyquist frequency, the lower the frequency of the contamination of the data [5, 6, 7, 8].

Now, suppose you're building an ADC system. The signals you're trying to transduce are between DC and

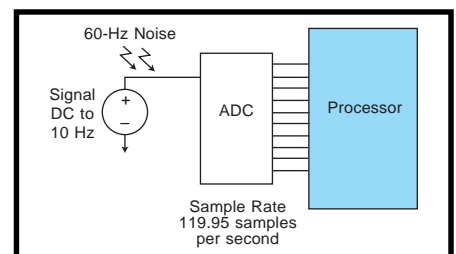


Figure 1—This hypothetical A/D system is set up with a 10-Hz signal and 60-Hz noise.

10 Hz in frequency. Your system is going to operate in a room brightly lit with fluorescent lamps, which are producing 60-Hz noise.

As the sample rate, let's choose 119.95 samples per second. What happens?

Figure 1 shows a block diagram of the system, and Figure 2 illustrates the noise and signal waveforms. The Nyquist frequency of the system is 59.975 Hz. The 60-Hz noise is undersampled and folds back.

The beat frequency between the sample rate and the noise (i.e.,  $60.000 - 59.975 = 0.025$  Hz) appears in the data. Because 0.025 Hz is in the frequency range you're transducing, you aren't able to distinguish between the aliased signal and a real signal, as shown by the graphs in Figure 3.

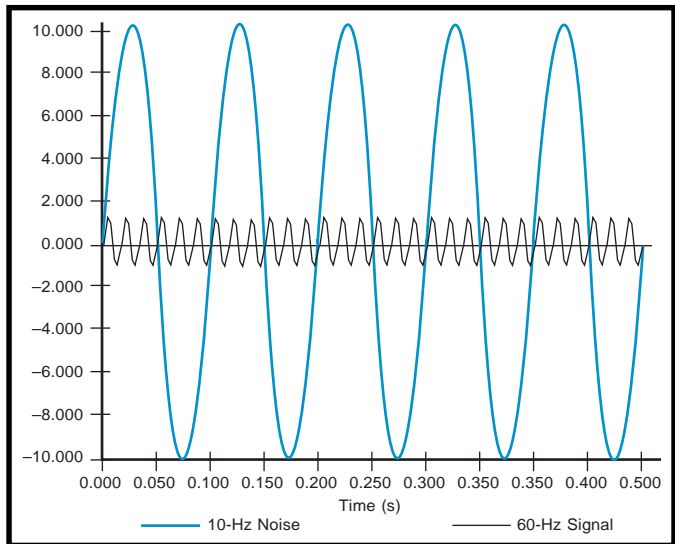


Figure 2—These waveforms (10-Hz input and 60-Hz noise) are input to the A/D system shown in Figure 1.

Now, this example may be contrived, but the problem is obvious. In the real world, the noise is never monotonic. The closer the noise frequency is to the Nyquist frequency, the lower the frequency of the aliased contamination.

## ALIASING AND SOFTWARE

Contrary to popular belief, it isn't possible to remove aliasing with software. Once the signal is acquired, there's no way to tease apart the real low-frequency components of the signal and any artifacts caused by aliasing [8].

Aliased contamination is sometimes masked by DC offset in the signal as well as genuine low-frequency noise. Keep in mind that, in a system with a sample rate of 10,000 samples per second, 60 Hz is low-frequency noise.

The only real solution to aliasing in a typical ADC system is a good old-fashioned low-pass analog filter [9]. The filter's cut-off frequency should be above the highest frequency to be transduced and below the Nyquist frequency of the system.

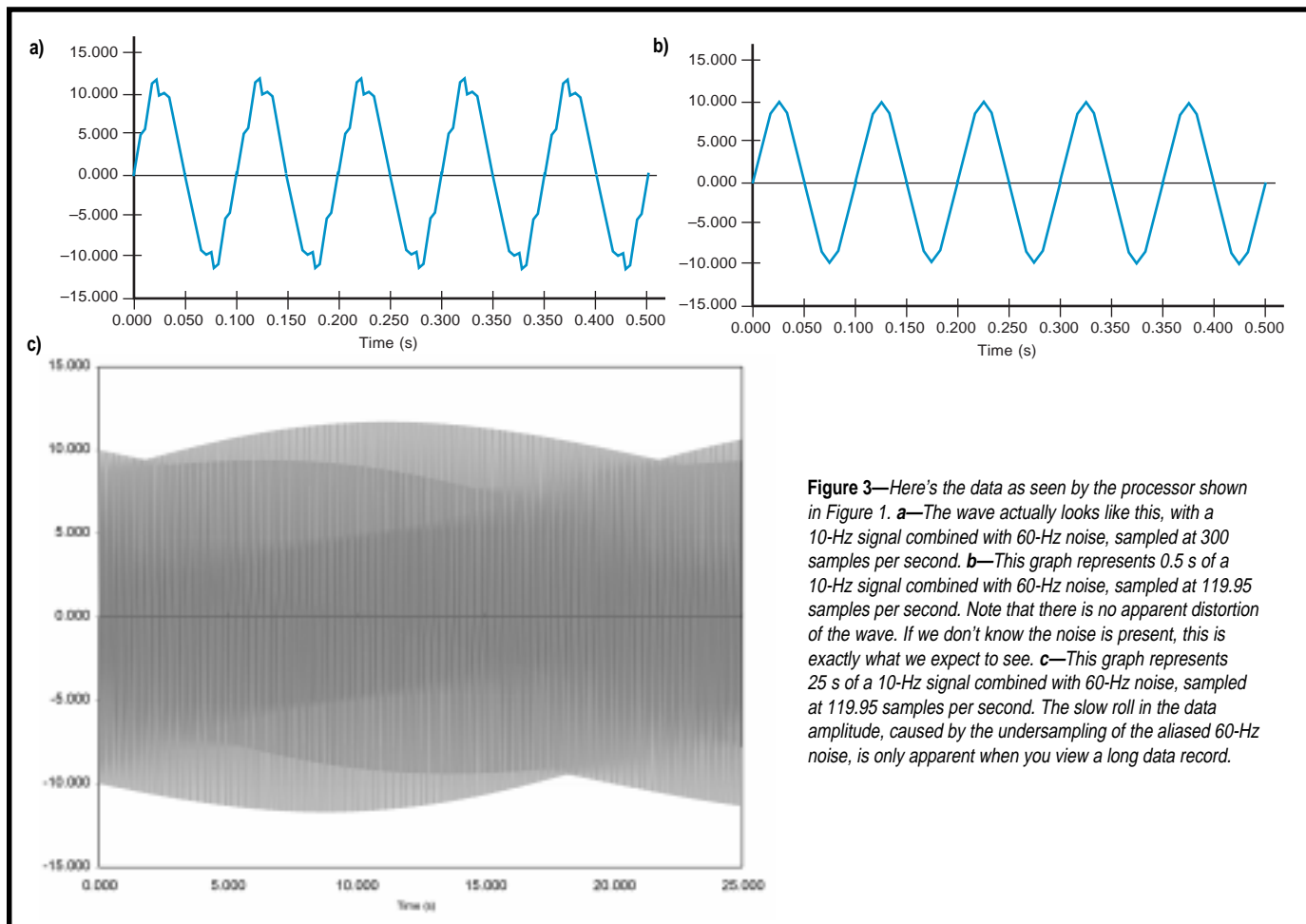


Figure 3—Here's the data as seen by the processor shown in Figure 1. **a**—The wave actually looks like this, with a 10-Hz signal combined with 60-Hz noise, sampled at 300 samples per second. **b**—This graph represents 0.5 s of a 10-Hz signal combined with 60-Hz noise, sampled at 119.95 samples per second. Note that there is no apparent distortion of the wave. If we don't know the noise is present, this is exactly what we expect to see. **c**—This graph represents 25 s of a 10-Hz signal combined with 60-Hz noise, sampled at 119.95 samples per second. The slow roll in the data amplitude, caused by the undersampling of the aliased 60-Hz noise, is only apparent when you view a long data record.

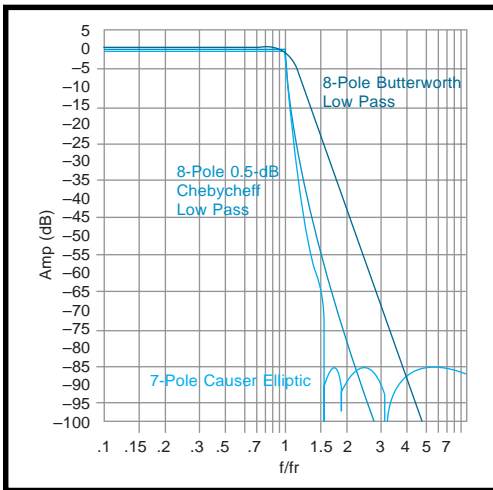


Figure 4—This graph compares typical low-pass filter transfer functions [10].

There are several approaches to this kind of filter problem. So, let's review them.

### PASS BANDS AND POLYNOMIALS

People have written volumes about which filter is the best. What's the bottom line? Go with the flattest pass band and the steepest cutoff at the Nyquist frequency that you can afford.

Of course, as with all things, there are tradeoffs. In most situations, a flat-pass band is more important than a steep cutoff. But, that's only true provided that there is enough cutoff to stop the aliasing.

Figure 4 shows pass- and stop-band characteristics of various filter types [10]. The corner frequencies of the anti-aliasing filter must be between the highest frequency of interest and the Nyquist frequency.

You want to place the anti-aliasing filter as close as possible to the input of the ADC. Also, the filter should have as many poles as possible, with eight being a general, practical limit for most systems. The aliased noise must be below the noise limit of the system at the Nyquist frequency.

If you observe the 10× rule [4], where the sample rate is chosen to be ten times the highest frequency of interest, then a Butterworth filter is a good all-around choice. If the highest

frequency of interest is closer to the Nyquist frequency, an elliptical filter provides a steeper rolloff. But then, you have to live with ripple in the pass and stop bands.

### ANTI\_ALIASING FILTERS

The simplest anti-aliasing filter is a capacitor between the signal and return. Strictly speaking, this one-pole RC filter is not the optimal solution (R is the output impedance of the signal source).

However, I think it's important to mention this option because there have been so many times I've been called in after a system was built and there was no hope of redesigning anything—and, naturally, the customer needed the system up and running right away. Hey, a capacitor is better than nothing.

So, use the reactance formulas, pick the best value, and solder it in. Cheap but effective, a well-placed capacitor can reduce aliasing and improve system performance. Sometimes, that's all you can do.

Passive filters introduce the least amount of extra noise into the signal, but there is insertion loss or attenuation of the signal's DC portion [6]. Passive filters are most appropriate for anti-aliasing where cut-off frequencies exceed 50 kHz or in situations where power is unavailable.

Active filters provide the best all-around solution because they're easy to implement. A variety of active solutions exist, and the filter can have gain greater than one.

### DO IT YOURSELF...

Generally, an ADC system has some type of amplifier sitting between the signal source and the ADC, and this amplifier can sometimes serve as a filter.

There are a lot of implementations of active filters using op-amps [1, 11]. Figure 5 shows a Sallen-Key implementation of a fourth-order Butterworth filter I've used with good results.

The trick to getting good results when building up your own active filters is to prototype and test. By building your own filter, you can save a lot in parts, but be sure to weigh

this against the added PCB real estate and added design time.

### ...OR OFF THE SHELF

Some excellent linear active-filter chips are on the market, one good example being the MAX275 from Maxim. As Figure 6 demonstrates, this eight-pole analog active filter can be configured for a variety of transfer functions. Maxim sells a DOS software package to help determine the proper value of passive components.

Using a linear filter chip can save design time and provide more consistent results. Another company with a great line of filters, Linear Technology, offers a Windows filter design program on CD-ROM.

### SWITCHED-CAPACITOR FILTERS

Another option—the switched-capacitor filter—is an active filter that uses electronic switching of a capacitor to imitate a high-order filter. Several manufacturers produce switched-capacitor filter chips.

The filter's cut-off frequency is controlled by a clock frequency applied to the chip, which controls the electronic switch. Typically, the clock frequency is 50–100 times the cutoff frequency of the filter [6, 11].

The major advantage of this setup is that the cutoff frequency is easy to change. So, if the system requires a variable sample rate, the antialiasing filter can follow along.

The major disadvantage is that a switched-capacitor filter is also subject to aliasing [6, 8, 11]. Eliminating the aliasing in the switched-capacitor filter sometimes requires a prefilter in front of the switched-capacitor filter and a reconstruction filter behind it. Some switched-capacitor filter chips have an op-amp linear filter onboard the chip to provide pre- or post-filtering.

Clock feedthrough, which is an extraneous signal that switched-capacitor filters create, can occur in the signal. This feedthrough resides at 50–100 times the filter's corner frequency. The presence of clock feedthrough can cause additional aliasing problems.

Switched-capacitor filter designs work best if the available space is small, there is ample time to debug

the design, and especially if the cutoff frequency has to be variable.

“Reducing Noise in a Switched-Capacitor Low-Pass Filter” is an excellent article detailing methods to better apply switched-capacitor filters [12]. Additionally, *The Art of Electronics* includes a detailed example system with a variable cut-off switched-capacitor antialiasing filter [11].

### FILTER MODULES

A filter module is a prebuilt filter unit that doesn't require any external components or adjustments. Filter modules are expensive, but the results are almost always good.

Two examples of filter modules are the D74 series and the 858 series from Frequency Devices. These are shown in Photo 1.

The D74 series are 16-pin DIP modules with a fixed frequency. The 858 series are programmable eight-pole active filters in a plastic module.

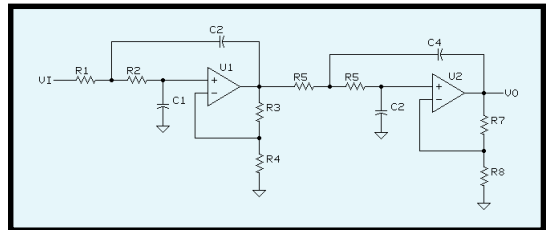


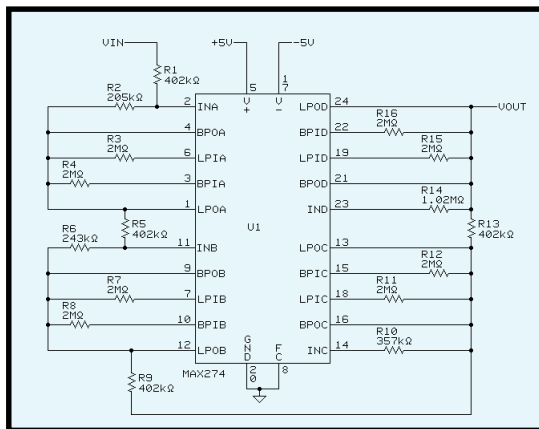
Figure 5—This four-pole Sallen-Key active filter can be built from standard op-amps.

These devices are available in several response curve formats and frequency ranges. Filter modules represent the easiest, most direct way out of an aliasing problem, but they certainly aren't the cheapest option.

### READING THE VOLTAGE

If the system in question just needs to read a voltage from time to time, your frequency of interest is 0 Hz (DC). This means that your sample rate should be at least  $2 \times 0 \text{ Hz} = 0 \text{ Hz}$ , which doesn't make sense until you realize that all frequencies are aliased in this situation.

Luckily, this is the one situation where software can help because any-



**Figure 6**—This eight-pole linear active filter ( $f_o = 1000$  Hz) was built with the MAX275.

thing that is not DC (i.e., anything that is moving) is bad.

An analog filter is still a good idea. But in addition to the filter, the ADC can be read several times in a row and all the results are then averaged. The number of samples to average depends on how much time is available, but a good rule of thumb is 100 samples [8].

### MULTIPLEXED INPUTS

Some systems have a multiplexed input to a single ADC. In fact, most off-the-shelf ADC boards fall into this category.

These systems come in two types—internal (including microcontrollers with multiple A/D inputs) and external. Figure 7 shows a multiplexed system with an antialiasing filter.

With an internal multiplexer, the point between the multiplexer's output and the input to the ADC is not accessible. Therefore, the only choice is to put an antialiasing filter on each channel.

However, in a system where the point between the output of the multiplexer and the input to the ADC is accessible, placement of the filter is not

so clear. A filter between the multiplexer and the input to the ADC is best if all the input signals are similar.

In a multiplexed system, the signal is sampled twice—once by the multiplexer and once by the ADC.

So, if you have an eight-channel multiplexer scanning eight signals, then by definition, the sample rate of the ADC must be at least eight times higher than the switching rate of the multiplexer. Therefore, the settling

time of your filter or amplifier must allow you to read the true signal [8].

### ALIASING'S ALIBI

Sometimes, however, aliasing isn't even the problem. A mechanical imperfection in the system can also produce alias-like effects. Unfortunately, a filter can't help you here.

Instead, consider a tachometer speed sensor with eccentric gears. A DC tachometer is mechanically connected to a shaft by a set of gears (see Figure 8). If the gears are not aligned with the center of the shaft, they will lobe (i.e., speed up and slow down) as the shaft spins.

The average velocity of the system is true, but the instantaneous acceleration—and thus the instantaneous velocity—is not constant. The output of the tachometer is a DC voltage with a small AC waveform on top of it, which results from the eccentricity of the gears.

This isn't aliasing in the classic sense. It's the tachometer reporting



**Photo 1**—These D74 and 858 series filter modules are manufactured by Frequency Devices.

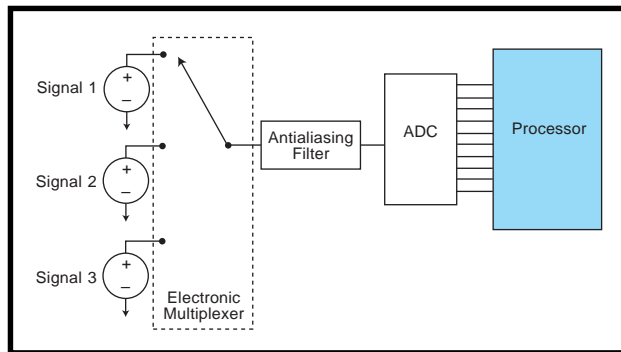


Figure 7—Antialiasing filters can also be used with multiplexed A/D systems.

the truth. I'll spare you a diatribe on my disdain for gears and just point out that the signal looks like it may have alias contamination, but in fact it does not.

Be on the lookout for such situations, especially where the sensors aren't directly coupled to whatever they're monitoring.

### RULES OF THUMB

With these rules of thumb, you should be able to focus your efforts whether you're designing a new system or troubleshooting an old one:

- If it is a sampled data system, aliasing can happen. Try not to hang around people who think that aliasing is something that only happens to Sigourney Weaver.
- When in doubt, filter it out. If you're designing a new system, plan on placing an analog low-pass filter as close as possible to the input of the ADC. The filter should have as many poles as you can afford, with eight poles being a general, practical limit for most systems.
- Cut off frequency. The filter's cut-off frequency should be below the Nyquist frequency and above the

highest frequency of the input signal.

- Many poles are good, but a cap to ground is better than nothing. Imagine that you're called in to "fix" a problem in an existing system and aliasing is the cause, but for some reason, you can't install a proper analog filter.

A properly chosen capacitor placed between the ADC input line and signal return provides a one-pole low-pass filter. This solution isn't the best, but it's usually easy to install, it reduces contamination, and you'll come out the hero.

- There is no antialiasing filtering in software—unless you have worked out the software implementation for the Houdini-Copperfield Mystery filter. Once the aliased low-frequency signal gets mixed with the real low-frequency signal, separation is impossible.
- Don't invite trouble. If possible, choose your sampling frequency to be something odd. Avoid choosing the frequency of the predominate noise source or its harmonics for the ADC sample rate.

In other words, avoid 60 Hz and harmonics of 60 Hz. (Believe me, I wouldn't include this rule here if I hadn't seen it done—more than once.)

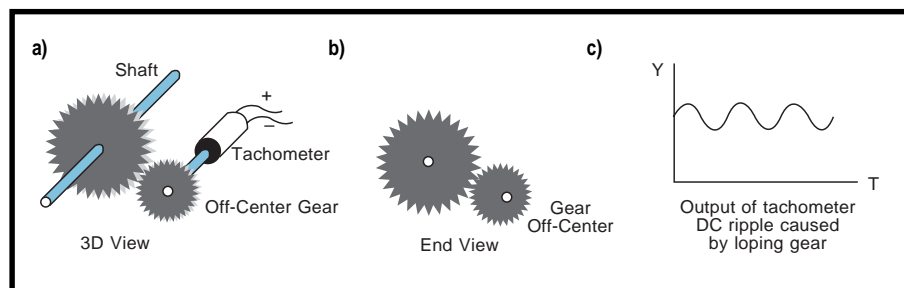


Figure 8a—This DC tachometer is connected mechanically to a shaft by a set of gears. The gear on the shaft of the tachometer is eccentric. b—This end view of the tachometer gear train shows the off-center tachometer gear. c—The output waveform with ripple is caused by gear eccentricity. If the gears are large enough, the ripple could be mistaken for aliasing.

- Know your noise. Use all the tricks—shielding, grounding, guard ring, and so on—you have at your disposal to keep the noise out.
- Understand system needs. Overkill is expensive. A design is a series of tradeoffs between conflicting goals.
- Look at the big picture. Aliasing most often appears as a slow DC roll. By looking at a long data record, you can see the periodic nature of the shift.
- Beware the finger-pointers. Make sure there isn't a mechanical or other defect that's producing an artifact that resembles aliasing.

Sure, aliasing can contaminate the data collected from an A/D system. But by applying a few rules, you can anticipate where and when aliasing problems may occur, and you'll have a good idea of the solution as well.

In the words of a wise friend of mine, *per ardua ad astra* (through difficulty to the stars). 🏠

*Thanks to Kathy Haynes for help with this manuscript. Most of all, I want to thank my father, J.M. Zerkus, for explaining the "wagon wheel" illusion to me.*

*Mike Zerkus is a results-oriented entrepreneurial engineer with 15 years' experience in instrumentation and control. Mike has worked in diverse areas, ranging from virtual reality and space shuttle payload development to wood products and biotechnology. He is the president of CM Research, a firm specializing in product development and scientific instrumentation. You may reach Mike at mzerkus@cmresearch.com.*

## REFERENCES

- [1] C. Watkins and S. Marenka, *Taking Flight*, M&T Books, New York, NY, 1994, 197–202.
- [2] H. M. Berlin, *Design of Active Filters with Experiments*, Howard W. Sams & Co., Indianapolis, IN, 1981.
- [3] C.T. Chen, *System and Signal Analysis*, Saunders College Publishing, New York, NY, 1989, p. 373.

- [4] D.-W. Jones, "Linear Control System Pitfalls," *Embedded Systems Conference*, Sept. 12–15, 1995.
- [5] R. Calkins, "For the Unwary, Aliasing Generates Problems in Unexpected Areas," *Personal Engineering & Instrumentation News*, July 1992, 47–51.
- [6] R. Steer, "Selecting Antialias Filters," *Sensors*, November 1992, 14–23.
- [7] S. Smith and C.-S. Pang, "Anti-aliasing Filters for Digital DA Systems," *Sensors*, November 1993, 30–33.
- [8] D. Potter, *Programmable Low-pass Filters for PC-Based Data Acquisition Boards*, National Instruments, App Note 058, 1995.
- [9] H.V. Malmstadt, C.G. Enke, and S.R. Crouch, *Electronics and Instrumentation for Scientists*, Benjamin/Cummings, Menlo Park, CA, 1981, 406–407, 412.
- [10] Frequency Devices, *Frequency Devices Design and Selection Guide*, Haverhill, MA, 1986, 3–29.
- [11] P. Horowitz, and H. Winfield, *The Art of Electronics*, Cambridge University Press, New York, NY, 1980, 1989, 281–284, 774–777.
- [12] R. Galter and V. Mayper, "Reducing Noise in a Switched-Capacitor Low-Pass Filter," *Sensors*, 13:12, p. 12, 1996.

## SOURCES

### MAX275

Maxim Integrated Products  
120 San Gabriel Dr.  
Sunnyvale, CA 94086  
(408) 737-7600  
Fax: (408) 737-7194

### D74 and 858 series filter modules

Frequency Devices  
25 Locust St.  
Haverhill, MA 01830  
(978) 374-0761  
Fax: (978) 521-1839  
www.freqdev.com

## I R S

404 Very Useful  
405 Moderately Useful  
406 Not Useful



# FEATURE ARTICLE

Tom Anderson

## Selecting Position Transducers

Are you choosing a position sensor or transducer but aren't sure how to weigh your options? Check Tom's parameter list. Prioritizing movement, environment, and contact requirements enables you to make the best decision.



As an application development manager for a position-transducer supplier, I get numerous queries on how to solve a broad range of position-measurement challenges.

These inquiries run the gamut from the common (aircraft flight-control surface movement) to the exotic (Formula One racecar suspension travel) to the seemingly impossible (three-dimensional tracking of a golf ball in flight from a fixed position).

These position-measurement challenges usually share one common element. They can be solved using a variety of solutions, but it's not always easy to determine the best one.

There are possibly more options for measuring position than any other type of sensed variable. While there may be more suppliers for pressure transducers, the variety of position-transducer types and technologies is unmatched.

The 1997 *Thomas Register* lists 264 suppliers of pressure transducers and 229 suppliers of displacement and position transducers. However, there are 13 categories related to displacement and position

measurement, compared to just four categories for pressure measurement.

In this article, I introduce you to various position-transducer selection parameters. You'll also find information on position-measurement techniques, technologies, and choices.

### BASIC TERMINOLOGY

But first, a brief note on semantics: for ease of communication, this guide refers to transducers and sensors as being the same. While not strictly true, it's not generally relevant whether you are using a position sensor or transducer. The goal of both is the same—to find out where something is!

Many kinds of transducers exist—pressure, temperature, velocity,.... Such devices provide position, displacement, and proximity measurements, which are defined as [1]:

- position—location of the object's coordinates with respect to a selected reference
- displacement—movement from one position to another for a specific distance or angle
- proximity—a critical distance signaled by an on/off output

In this article, I focus primarily on transducers for position and displacement measurement. And unless otherwise noted, I use the term "position transducer" to refer to displacement and proximity transducers as well.

### THE PARAMETERS

On what basis should you select a position transducer? As a starting point, let's look at the laundry list of



Photo 1—Cable position transducers provide extended ranges in small sizes. Flexible cable allows for easy installation.

parameters shown in Figure 1. While this list is not all-inclusive, it helps you begin to decide what parameters are relevant to your application.

Perhaps the first parameter to address in any application is whether the transducer can physically touch the object being monitored. If your application is sensitive to outside influences, a noncontact transducer may be the most appropriate. Otherwise, a contact sensor might offer advantages not found in a noncontact sensor.

At first thought, noncontact transducers may seem like the superior solution for all applications. However, the decision isn't that clear cut.

Noncontact products can emit potentially harmful laser- or ultrasonic-based signals. These products also rely on having a clear visual environment to operate in. Frequency response isn't always as high as with a contact sensor, but costs are often higher. Finally, operating-temperature ranges are typically not as broad.

Another parameter to consider early on is whether you need to measure linear or rotary movement. Note that using cable position transducers (like the one shown in Photo 1), cams,

pulleys, levers, electronics, software, and other methods can enable a rotary transducer to measure linear motion, and vice versa. Lack of space, cost, and ease of mounting are a few reasons for doing this.

Once you decide if you require a contact or noncontact solution and are measuring rotary or linear movement, selecting a transducer technology becomes much easier.

Next, determine if you're monitoring one-dimensional or multidimensional motion. If the motion is multidimensional, see if you need to measure in multiple dimensions or if the object is moving in multiple dimensions and you only have to measure one of them. Often, multidimensional motion is measured with multiple one-dimensional transducers.

Also, think about the type of signal you need to obtain. If you need a signal that specifies a unique position, be sure to specify a transducer with absolute output.

However, if all you need is relative position from a prior position or a simple on/off indicator, then incremental or threshold technology is more appropriate. Photo 2 gives you a

view of some incremental rotary optical encoders.

An important difference between incremental and absolute transducers is that incremental transducers typically need to be reinitialized after power-down by moving the monitored object to a home position at powerup. This limitation is unacceptable in some applications.

Threshold measurements are on/off in nature and usually involve limit switches or similar devices. As you might guess, absolute devices are usually more expensive than incremental or threshold devices.

Travel, also known as range, varies from microns to hundreds of feet (or more, depending on your definition of "transducer"). The range of many precision transducers is limited to 10' or less.

If your application needs to operate on the Space Station or some other size- and weight-sensitive platform, you need to specify the maximum values for the transducer's dimensions and weight.

The application's operating environment can have a large impact on your technology choice as well. You need

to determine what operating and storage temperatures the device will be in and whether you need to meet commercial, industrial, or military environmental requirements.

Also consider whether excessive humidity, moisture, shock, vibration, or EMF will be encountered. See if your environment has other unique aspects, such as high or low pressure or the presence of hazardous or corrosive chemicals.

An often-overlooked parameter is the method and time required for transducer installation and mounting. For

Parameter	Relevant?	Ranking	Choices
Contact	<input type="checkbox"/> Yes <input type="checkbox"/> No		<input type="checkbox"/> Contact <input type="checkbox"/> Noncontact
Motion Type	<input type="checkbox"/> Yes <input type="checkbox"/> No		<input type="checkbox"/> Linear <input type="checkbox"/> Rotary
Dimensions	<input type="checkbox"/> Yes <input type="checkbox"/> No		<input type="checkbox"/> One Dimensional <input type="checkbox"/> Multidimensional
Measurement Type	<input type="checkbox"/> Yes <input type="checkbox"/> No		<input type="checkbox"/> Absolute <input type="checkbox"/> Incremental <input type="checkbox"/> Threshold (Proximity)
Range	<input type="checkbox"/> Yes <input type="checkbox"/> No		<input type="checkbox"/> Less than 1" <input type="checkbox"/> 1-30" <input type="checkbox"/> Greater than 30"
Physical Size/Weight	<input type="checkbox"/> Yes <input type="checkbox"/> No		<input type="checkbox"/> Size Restriction_____ <input type="checkbox"/> Weight Restriction_____
Environmental	<input type="checkbox"/> Yes <input type="checkbox"/> No		<input type="checkbox"/> Humidity <input type="checkbox"/> Vibration <input type="checkbox"/> Corrosion <input type="checkbox"/> Moisture <input type="checkbox"/> Temperature <input type="checkbox"/> Other_____
Installation/Mounting	<input type="checkbox"/> Yes <input type="checkbox"/> No		<input type="checkbox"/> Removable <input type="checkbox"/> Installation <input type="checkbox"/> Time Limit _____
Accuracy	<input type="checkbox"/> Yes <input type="checkbox"/> No		<input type="checkbox"/> Linearity <input type="checkbox"/> Resolution <input type="checkbox"/> Repeatability <input type="checkbox"/> Hysteresis
Lifetime	<input type="checkbox"/> Yes <input type="checkbox"/> No		<input type="checkbox"/> Cycles_____ <input type="checkbox"/> Hours of Continuous Operation_____
Cost	<input type="checkbox"/> Yes <input type="checkbox"/> No		<input type="checkbox"/> Less than \$50 <input type="checkbox"/> \$50-\$500 <input type="checkbox"/> Greater than \$500
Delivery	<input type="checkbox"/> Yes <input type="checkbox"/> No		<input type="checkbox"/> Less than 1 Week <input type="checkbox"/> 1-4 Weeks <input type="checkbox"/> Greater than 4 Weeks
Output	<input type="checkbox"/> Yes <input type="checkbox"/> No		<input type="checkbox"/> Analog Voltage <input type="checkbox"/> Analog Current <input type="checkbox"/> Digital <input type="checkbox"/> Sensor Bus_____ <input type="checkbox"/> Visual <input type="checkbox"/> Other_____
Frequency Response	<input type="checkbox"/> Yes <input type="checkbox"/> No		<input type="checkbox"/> Less than 5 Hz <input type="checkbox"/> 5-50 Hz <input type="checkbox"/> Greater than 50 Hz

Figure 1—What are your requirements? This table helps you rank your most important parameters and value specifications. Select the relevant parameters, prioritize them, and then choose the appropriate value for the parameters.

testing applications, this parameter may not be so important. However, OEM and large-volume applications often require simple installation and removal to reduce labor costs and enable easy maintenance.

See if the transducer can only be mounted with manufacturer-provided special mounting bases or if a variety of mounting techniques can be used. Besides the common threaded-fastener approach, some other nonpermanent mounting techniques include suction cups, magnets, industrial adhesives, grooved fittings, and clamping.

In going through the previous parameters, you might have asked yourself, "Hey, what about accuracy?" While accuracy is certainly important and sometimes critical, it's often the last degree of freedom in the selection of a transducer.

As you may know from experience, accuracy is not a well-agreed-on term. Typically, various components of accuracy—linearity, repeatability, resolution, and hysteresis—are quoted for vendor convenience or per user requirements.

With the availability of software calibration tools today, linearity isn't as important as it once was. For many applications, in fact, repeatability is the most important component.

Accuracy is typically specified in absolute units like mils or microns or in relative units such as percent of full-scale measurement. If you are comparing the accuracy of one device against another, make sure you are comparing apples to apples.

For example, see if the accuracies being quoted are at a single temperature or over a temperature range. If you need it, find out if temperature compensation is available.

If you expect to see significant numbers of cycles or if the transducer will be in service for an extended period of time, specify the lifetime and reliability requirements as well. When choosing the transducer, find out what warranties are offered as well as how maintenance and repairs are handled.

A transducer that can be repaired in-house can reduce costs significantly. You should also consider what type of periodic recalibration is recommended and whether calibration procedures are provided.

It's a good idea to ask vendors what type of use their transducers see most often. Common uses include OEM, retrofit, industrial control, commercial, and test and measurement. Hopefully, the transducer has seen previous use in your type of application.

In the early stages of transducer specification, product cost sometimes doesn't even make the list. More often than not, this parameter gains importance as the project moves forward.

When you're determining costs, make sure to look at the initial acquisition cost as well as the cost over the product's life. For example, are special signal-conditioning electronics, power supplies, electrical connectors, housings, installation tools, or mounting fixtures required?

Ask the vendor for typical repair, maintenance, and replacement costs. And, inquire about the cost of the transducer in volume and single-unit quantities. The cost savings (e.g., a cost of \$100 in volume but \$600 in single quantities) may be an important factor if small-quantity replacement units will be needed in the future.

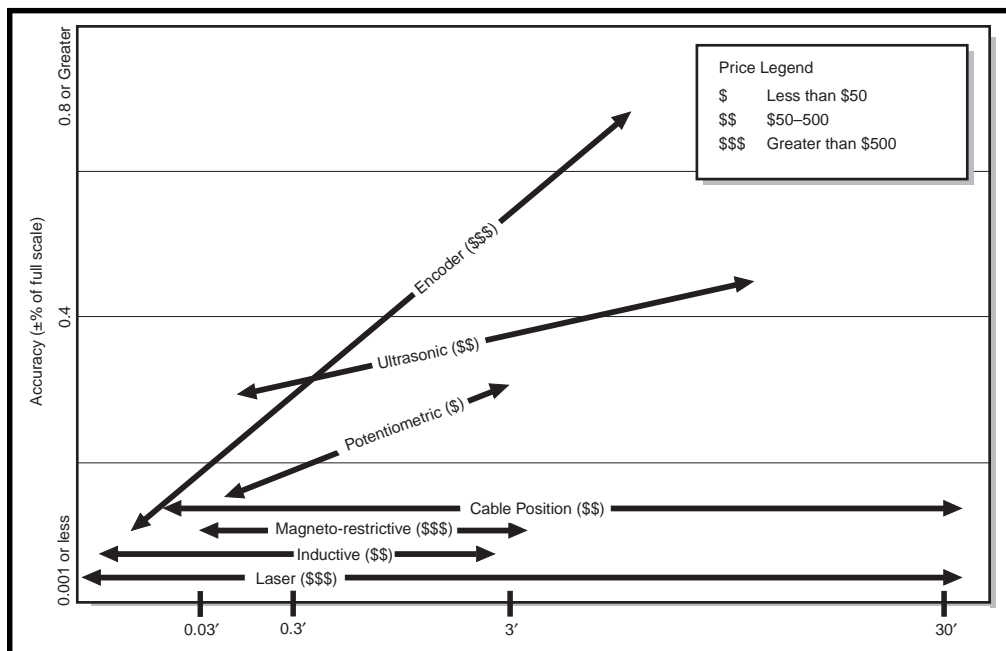
Another parameter that's occasionally overlooked is the time it takes the product to be delivered to you after you order it. The custom nature of some transducers combined with production processes and manufacturing economics requires lead times of eight weeks or more.

This delivery schedule might be acceptable now, but what about in six months when you need extra quantities or a spare part? Evaluate whether or not you can afford to be without a part for an extended period of time.

Obviously, the transducer is going to be a part of a system. So, determine your preferred electrical input and output requirements. Common output choices include analog AC and DC voltage, resistive, current (4–20 mA), digital, and visual (meter).

Increasingly, outputs using sensor bus protocols are being offered. Most position transducers require 50 V or less, and some are self-powered.

Finally, for fast-moving applications, determine the maximum velocity or acceleration that needs to be monitored. Ensure that your



**Figure 2**—It's true: you can't have it all. As with many specification decisions, tradeoffs must be made when you're selecting a position transducer. This graph shows the typical performance of some linear position transducers as compared by maximum range, best accuracy, and cost.

data-acquisition or control system has an adequate sampling rate to record the resulting datastream.

## CHECK YOUR REQUIREMENTS

Now that you're aware of the key parameters, you need to determine which ones are relevant to your application and of these relevant parameters, which are most critical.

If you don't prioritize your requirements, it's going to be difficult to make a selection decision. You may come to the conclusion that there is no transducer that can meet your needs. This may be true, but it's more likely that your requirements are too stringent and that you need to make a tradeoff to arrive at the optimum selection.

For example, an engineer recently approached our company looking for a transducer with  $\pm 0.0001''$  resolution over 30', and he wanted to keep the



**Photo 2**—These incremental rotary optical encoders provide quadrature digital output. Encoder use is increasing as more transducers are being connected directly to digital processing systems.

cost under \$500. He was adamant that all three specifications be met. Our products didn't meet all his specifications, and we were at a loss as to where we would refer him.

After some more discussion, we found out that the resolution requirement was only necessary over a limited portion of the total range and that the cost goal, while important, did have some flexibility. Hence, in this situation, range was most important, followed by resolution, and then cost.

The moral of this story: focus on your top requirements. Make the best

decision you can, given the specifications you need. And keep in mind, you can't have everything—unfortunately.

## NEXT STEPS

In this article, I've given you some parameters for selecting position transducers. But in case you hadn't

noticed, I didn't provide any information on what type of technology you should select for your position transducer.

The constant change in transducer technology and the difficulty in generalizing about a particular technology's capabilities and limitations mean there's no way I can cover this area in detail here.

Additionally, choosing the technology should come after determining and prioritizing your requirements. Once your requirements are well-known, the choice of technology tends to be self-selecting.



Photo 3—Laser position sensors have resolutions of 0.1  $\mu\text{m}$  or better.

For example, just knowing whether you require a contact or noncontact technology can cut your choices almost in half. If you need the latter, a laser position sensor like the one in Photo 3 may be a good choice.

To get a feel for the capabilities of some of the more prevalent linear position-measurement technologies, Figure 2 maps out how these technologies compare against each other based on cost, accuracy, and maximum range. Note that not all technologies are shown.

Now, it may be difficult to clearly define the parameter values you require as well as which parameters are most important in your application. However, it can be even more difficult to obtain these parameters from vendors and then compare one vendor's statements against another's.

To get information on products beyond what you see in the vendor's product literature, review transducer-related publications such as *Measurements & Control* and *Sensors* for articles on position-measurement products and technologies.

Also, be sure to ask your colleagues about their experiences and recommendations. They may have a position transducer on hand that you may be able to test for your application.

Of course, in this day and age, make an effort to search Web engines and Internet newsgroups. Numerous engineering, instrumentation, and measurement-oriented newsgroups can be reached via search engines. Extensive sources of position-trans-

ducer manufacturers can be found in the *Thomas Register* and the *Sensors Buyer's Guide*.

Contact vendors and request references of similar applications. Ask these references why they selected the product they did and whether they're happy with their decision. Also, find out what other options they considered.

Finally, see if the vendor has product samples or evaluation units you can use for testing before purchase. If the vendor is hesitant to do this, offer to provide them with a test report summarizing your evaluation. This information may be valuable to them, and they may be more willing to assist you. ☒

*Photos 1, 2, and 3 are courtesy of SpaceAge Control, OakGrigsby, and Dynamic Control Systems, respectively.*

*Tom Anderson is application development manager at SpaceAge Control, a manufacturer of miniature and sub-miniature position transducers and flight test air data products. For over 10 years, he has been involved in computer and instrumentation product design and development at Pacific Bell, Apple Computer, and Hewlett-Packard. You may reach him via email@spaceagecontrol.com.*

## REFERENCES

- [1] J. Fraden, *AIP Handbook of Modern Sensors*, American Institute of Physics, New York, NY, p. 264, 1993, 1996.

## RESOURCES

### Texts

- Schaevitz Engineering, *Handbook of Measurement and Control*, Pennsauken, NJ, 1976.
- I. Busch-Vishniac, *Electromechanical Sensors and Actuators*, Springer-Verlag, New York, NY, 1998.

*Thomas Register Directory of American Manufacturers*, Thomas Publishing Co., New York, NY, 1997.

### Internet

*Sensor's Buyer's Guide*,  
www.sensorsmag.com  
*Thomas Register*,  
www.thomasregister.com

## SOURCES

### Sensors

Dynamic Control Systems  
7088 Venture St., Ste. 205  
Delta, BC

Canada V4G 1H5  
(604) 940-0141  
Fax: (604) 940-0793  
www.dynavision.com

MicroStrain, Inc.  
294 N. Winooski Ave.  
Burlington, VT 05401  
(802) 862-6629  
Fax: (802) 863-4093  
www.microstrain.com

Midori America  
2555 E. Chapman Ave., Ste. 400  
Fullerton, CA 92831  
(714) 449-0997  
Fax: (714) 449-0139  
www.thomasregister.com/midori

OakGrigsby, Inc.  
84 N. Dugan Rd.  
Sugar Grove, IL 60554  
(630) 556-4200  
Fax: (630) 556-4216  
www.oakgrigsby.com

Senix Corp.  
52 Maple St.  
Bristol, VT 05443  
(802) 453-5522  
Fax: (802) 453-2549  
www.senix.com

SpaceAge Control, Inc.  
38850 20th St. E  
Palmdale, CA 93550  
(805) 273-3000  
Fax: (805) 273-4240  
www.spaceagecontrol.com

## IRS

407 Very Useful  
408 Moderately Useful  
409 Not Useful

**41** Nouveau PC  
edited by Harv Weiner

**46** Real-Time PC  
Graphical User Interfaces  
in RTOSs  
Ingo Cyliax

**54** Applied PCs  
A New View  
Part 1: Virtual  
Instrumentation  
Fred Eady



Photo courtesy of  
National Instruments

## SCALABLE CPU

The **Complete Scalable CPU** module is a highly integrated, scalable, PC/AT platform that conforms to the PC/104 standard. Design options, using the CardIO credit-card-sized CPU series by S-MOS, include an 8086, 80386, or 80486 processor that features IBM-PC-compatible functions.

The module accommodates up to 16-MB RAM, operates at clock speeds up to 100 MHz, and comes in 8- and 16-bit bus versions. Applications include remote monitors, ATMs, medical devices, factory automation, and telecommunications.

The Scalable CPU features two serial ports, one parallel port, LCD/CRT controller, IDE and floppy disk controllers, keyboard/mouse controller, and IBM-compatible BIOS. It is available with PC/104 industry-standard power and utility connections, and it offers separate connections for reset, speaker, and external battery. The unit uses



RS-232 voltage generation, operates from one +5-VDC supply, and features low power consumption. A watchdog timer and a socket for M-Systems' flash DiskOnChip are also included.

The Scalable CPU is compatible with industry-standard software applications and development tools such as IBM PC-DOS, Microsoft MS-DOS, Windows, Windows 95, Pen OSs, Visual Basic, Visual C++, and real-time operating systems like QNX.

Pricing for the Scalable CPU starts at **\$199** (without a flash module).

**The parvus Corp.**  
**396 W. Ironwood Dr.**  
**Salt Lake City, UT 84115**  
**(801) 483-1533**  
**Fax: (801) 483-1523**  
**www.parvus.com**

#510

## WEB SOFTWARE DEVELOPMENT KIT

The **Voyager Software Development Kit (SDK)** enables QNX developers to build full-featured Internet applications for the embedded marketplace. It contains all the industry-standard tools and code needed to develop a complete, customized Web browser. The SDK includes the source code for the Voyager browser interface, source code for E-mail, Internet dialer and newsreader applications, a demo with source code for controlling



the browser with a hand-held remote, configuration files for the Photon Application Builder, and a Widget Library.

Voyager is based on the QNX real-time operating system and Photon microGUI (an embeddable windowing system). These systems, along with the Voyager browser, TCP/IP stack, and Internet dialer are functional in under 2-MB flash memory and 40-MB RAM. Voyager SDK supports all the current Internet standards, including HTML 3.2, frames, tables, proxy servers, SOCKS, server push, client pull, progressive image display, animated GIFs, JPEGs, FTP, basic and digest authentication, printing gopher, cookies, helpers, plug-ins, CGI, POP3, SMTP, and more. Voyager provides a command-driven interface so the browser can be controlled from any input device.

Pricing for the Voyager SDK, including source code, is **\$10,000**. OEM run-time pricing is also available.

**QNX Software Systems, Ltd.**  
**175 Terence Matthews Cres.**  
**Kanata, ON**  
**Canada K2M 1W8**  
**(613) 591-0931**  
**Fax: (613) 591-3579**  
**www.qnx.com**

#511

# Nouveau PC

edited by Harv Weiner

## DATA-CAPTURE SOFTWARE

**WinWedge 32 Pro** adds serial data-acquisition and instrument control to any Windows, Windows 95, or Windows NT applications. It automatically collects data from instruments or serial devices such as lab equipment, pH meters, balances, moisture analyzers, bar-code readers and scanners, titrators, spectrometers, ion meters, radiation counters, data loggers, and modems.

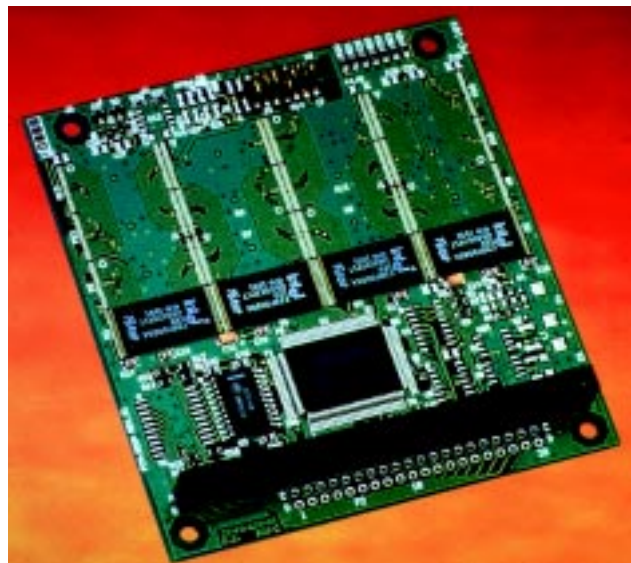
Real-time analysis, charting, and graphing of your instrument data in popular PC applications can be easily performed. Different instruments can simultaneously send data to different applications or to different fields within the same application.

TCPWedge is now included at no additional charge. TCPWedge enables the user to communicate with any TCP/IP address over any network directly from any Windows application. So, data can be collected directly from instruments linked to a PC's serial ports via serial cables (up to 100 devices simultaneously on one PC), over an internal TCP/IP network (i.e., Ethernet) from any IP address on the network, or over the Internet from any IP address on the Internet, even at remote sites. WinWedge 32 Pro can send data and commands out through serial ports to devices, and TCP/Wedge can send data out over networks to any IP addresses.

WinWedge 32 Pro sells for **\$495**. TAL Technologies also offers other versions of this software priced between **\$199** and **\$395**.

**TAL Technologies, Inc.**  
**2027 Wallace St. • Philadelphia, PA 19130**  
**(215) 763-7900 • Fax: (215) 763-9711**  
**www.taltech.com**

#512



## PC/104 FLASH MODULE

The **PCM-3820** is a high-density PC/104 flash module that uses the latest TSOP flash-memory chips for diskless nonvolatile storage. This bootable, high-speed storage module allows programs and data to be read and written (executed and copied) just like a standard disk drive but with extremely fast access speed and the reliability of no moving parts. It is compatible with a wide range of operating systems. Also, the included TrueFFS software makes it a simple to use, powerful solution for embedded data storage.

The PCM-3820 features a transfer (read) rate of up to 2 MBps and an average seek time less than 0.1 ms. Its MTBF is greater than 1 million hours, and its data retention is 10 years minimum without a power source. It also features unlimited read cycles and a minimum of 100,000 write cycles. The module can be ordered with 1–32 MB of storage space, and up to 128 MB (four cards) can be used per system.

The 4-MB version of the PCM-3820 is priced at **\$127** in OEM quantities.

**VersaLogic Corp.**  
**3888 Stewart Rd.**  
**Eugene, OR 97402**  
**(541) 485-8575**  
**Fax (541) 485-5712**  
**www.versalogic.com**

#513

*Nouveau* PC



## PC-BASED CONTROLLER

The **ADAM-4500** is a fully functional IBM-compatible PC in a small package. Its built-in ROM-DOS is an MS-DOS-equivalent operating system, providing the basic functions of MS-DOS except for the BIOS. As a result, the module can run standard PC software written in a high-level language like C or C++. The ADAM-4500 can relieve the host computer of many controller functions, enhancing overall system performance. Additionally, it also permits the automation system to be isolated from the host controller.

The unit features an 80188 CPU, 256-KB flash memory, 256-KB SRAM, two serial ports (COM1 and COM2), real-time clock, watchdog timer, and a download port. The COM1 port can be configured for either RS-232 or RS-485, while COM2 is a dedicated RS-485 port. Also, 170 KB of flash memory is available to the user for installing applications via the download port. An additional 234-KB RAM is

available for running applications or for string data or results. An installation utility helps users transfer programs from the development PC, since applications must be converted into 80188-compatible code before they can be installed and run on the ADAM-4500.

The ADAM-4500 is priced at **\$350** in single quantities.



**Advantech America**  
**750 E. Arques Ave.**  
**Sunnyvale, CA 94086**  
**(408) 245-6678**  
**Fax: (408) 245-5678**  
**www.advantech-usa.com**

#514

# Nouveau PC

Ingo Cyliax

# Graphical User Interfaces in RTOSs

*We can interface with an embedded PC via the Web, but how about a view that doesn't depend on HTML? And, what counts as real time for GUIs, anyway? Ingo has the answers as he gives us a close-up look at X Windows and QNX's Photon.*

Often, your embedded system needs some kind of GUI. This month, I want to take a close look at two popular GUI systems for embedded real-time PCs—X Windows and QNX's Photon microGUI.

X Windows wasn't originally developed for embedded systems, but it has become a popular GUI system that's available under several RTOSs. And, it's freely available on the Internet, making it a portable GUI programming environment.

Photon, on the other hand, is a proprietary GUI system developed by QNX. It offers advantages such as a small footprint and QNX's message-passing interface.

I'll briefly contrast these interfaces to Web-based GUIs, but I won't go into detail about Web-based GUIs because they've received a lot of coverage lately ("Remote Internet Data Logging and Sensing," *INK* 88, "Interfaces and GUI-Building

Packages," *INK* 88 and 89, "Converting PC GUIs for NonPC Devices," *INK* 91).

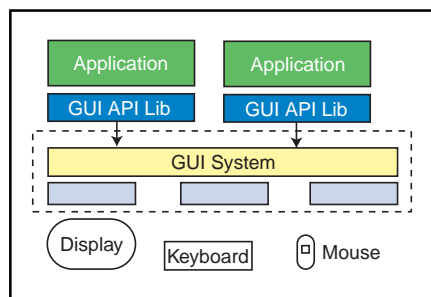
When looking at GUIs for embedded PC RTOSs, we need to consider memory requirements and real-time response issues, as well as how distributed the GUI system is. We also have to worry about hardware compatibility with the PC.

Of course, memory requirements depend on the application. Some GUIs are fairly small and can be ROMed. However, they can be limited in functionality. But if a small memory footprint is required, that tradeoff is probably OK.

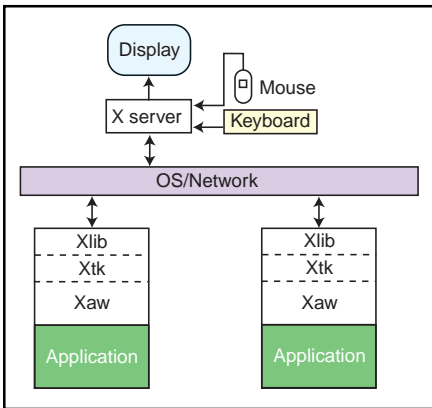
X Windows, for example, is scalable. Its display engine can be fairly small, and applications have the choice of libraries to make the GUI either Spartan or flashy, which is likely to enlarge the application.

Real-time response is probably harder to nail down. As a user, I'm concerned about how the user interface responds to inputs. For example, if I click on a pop-up menu, I expect it to pop up right away.

RTOS developers know that this kind of real-time response is just soft real time. Most users probably can't tell the difference in feedback if the response varies between 10 and 30 ms. In fact, any



**Figure 1—Here we have a monolithic GUI architecture. The application and the GUI system are linked into one object.**



**Figure 2—The X Windows architecture separates the application (i.e., the X clients) from the device drivers (i.e., the X server), enabling X applications to be distributed.**

response below 30 ms seems instantaneous. Contrast this with hard real-time responses, where you want consistent and deterministic responses to events like interrupts.

A real-time system designer should be more concerned about how the GUI affects the deterministic behavior of the whole system. This issue may be problematic if having a GUI subsystem adds significant uncertainty to the interrupt response.

One way to address real-time issues with GUIs is to decouple the GUI, which is soft real-time, from the processes in the system that need deterministic hard real-time responses. This separation is achieved by putting the GUI in separate tasks that run at lower priority than time-critical tasks.

Many GUIs take this concept a step further by separating application-specific code from the rendering and frame-buffer device-driver code. In X Windows, the X application code is called the X client, and the code driving the hardware is the X server. Photon also follows this model.

Client- and server-based GUI systems enable the design of distributed systems. For example, the server may be on one node, driving the display hardware, and several client applications then run on other nodes networked together. Contrast this setup to a monolithic GUI system, where the GUI and device driver are part of the same program on the same node.

Finally, GUIs on embedded real-time PCs present an extra twist. While running GUI-based OSs like Windows on PCs using VGA displays is commonplace, it's easy to forget VGA cards vary quite a bit in their implementations.

VGA chipsets are programmed differently at the low level to take advantage of

special features. Under Windows, you can use the card vendor's device drivers to abstract these differences and present a consistent API to the OS.

However, card vendors typically don't provide device drivers for RTOSs, so it's on the RTOS vendor's shoulders to develop device drivers for various cards. X Windows' popularity is probably due to many SVGA/VGA cards being supported by the freely available distribution of the X Windows system, XFree86.

## DISPLAY AND INPUT DEVICES

With GUIs, three devices are of particular interest—the display, keyboards, and pointing interfaces. I've already talked a little about the display device, which is typically standard VGA or SVGA cards for PCI, Vesabus, or ISA bus.

In the embedded world, however, you find VGA/SVGA-based cards for embedded buses like PC/104, PC/104+, and STD-32. These cards are based on chipsets like those from Western Digital and Cirrus. The chipsets support nonaccelerated or accelerated operations, and some offer 2D and 3D rendering/acceleration.

I already mentioned the importance of making sure your RTOS vendors support the GUI interface for the chipset on the card you choose. But if you plan to use the standard VGA resolution of 640 × 480 and 16 colors, don't worry. Almost all cards are compatible in this mode.

The trouble comes when you want to use an SVGA mode like 1024 × 768 or extended color space. If the card is accelerated, make sure the device driver supports this functionality. Otherwise, just save the money and buy a cheaper nonaccelerated card.

Vendors that sell SVGA cards for embedded systems buses like PC/104 and STD-32 include Versallogic (which supports PC/104+), Ampro, Real Time Devices, and WinSystems. These cards may be supported by some RTOS vendors, but check them out to make sure.

Many display cards especially targeted for the embedded-PC market also

have a LCD interface.

The LCD interface isn't as standardized as the VGA interface for CRTs, so you have to ensure your display card also supports the LCD panel in your system. The LCD backlights require a high-voltage power supply, which isn't included on most LCD-capable display cards.

Keyboards are much easier on PCs. They come in AT (5-pin DIN) or PS/2 (mini-DIN) varieties.

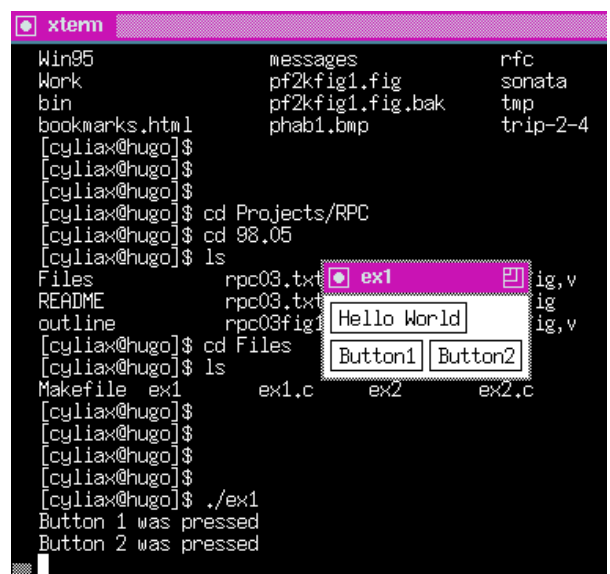
The keyboard interface is handled by the keyboard controller and presents a standard interface to the system, so support usually isn't a problem. A keyboard is a good way to interface external devices to the system.

Some available devices include mini-keyboards or special boxes that generate preprogrammed key sequences depending on an external stimulus. Bar-code and card scanners can also be made to talk to the system via the keyboard interface, by using an RS-232-to-keyboard converter.

Pointing devices may be the most interesting interfaces. We're all used to the standard mouse, which works through a standard serial port (COM port) or PS/2 interface. There's also the bus mouse, which uses a special card.

Mice use a variety of protocols. Logitech, Microsoft, and Mouse Systems are some of the more popular brands.

Other pointing devices include track balls and touch-sensitive screens. Touch-



**Photo 1—Here's what our example GUI looks like under X Windows. The Athena widget library I used fits in with the look and feel implemented by the tiled window manager (TWM), which handles all the decorations (e.g., title bars).**

sensitive screens are film based and overlay the display device. The coordinates where the user touches the film are sent to the system. There are also wireless tracking devices, like 3D mice and pointing devices, that interface via serial ports or PS/2 interfaces.

While most pointing and tracking devices emulate popular mouse protocols and ease interfacing to your GUI systems, some do not. So, just like the display device, make sure the pointing device is supported by the GUI.

## TECHNIQUES

There are two basic strategies for GUI systems—monolithic and distributed.

In a monolithic system, as diagrammed in Figure 1, the GUI, GUI-support libraries, and possibly even the device driver are linked into the application. This approach is good if you want to minimize memory requirements and achieve the highest performance of your graphics subsystem.

The distributed method separates the application from the device driver and display system. It uses remote procedure calls or message passing to communicate between them. Figure 2 illustrates one such implementation, X Windows.

X Windows has two major components—the X server and the X client. The X server contains the device-dependent drivers for the display, keyboard, and pointing devices. Its device-independent driver element abstracts the differences in devices to present a more consistent interface.

At the other end, the X protocol interface implements an RPC-based protocol. The X protocol runs on many network implementations, but typically it runs over TCP/IP or whatever message-passing mechanism the OS implements for local-node communications.

Unlike OSs such as MS Windows, X Windows doesn't prescribe the look and feel. Each X client, however, controls how its windows appear and how events from the keyboard and pointing device are handled.

An X client connects to the X server and typically creates a window on the display. Windows from different clients can overlap.

The events from the keyboard and pointing device are sent to the clients that are interested in the events. The client calls primitives on the X server via its RPC protocol to render bitmaps or characters

using fonts. To enhance performance, the X client stores fonts, bitmaps, and color mappings in the X server.

You can use a special X client—the window manager—to implement the system's desktop. X Windows even lets you implement a Windows 95-like desktop!

Programming using the X protocol directly, however, is tedious. So, X Windows features a toolkit for developing widgets.

Widgets define a windows object, like a button, and handle the details of managing it. For example, a button-widget handles drawing and redrawing itself.

**Listing 1—This program illustrates the basics of a widget-based X application. The developer initializes the widgets and calls `MainLoop` to handle event processing and call-back dispatching.**

```
#include <stdlib.h>
#include <X11/Xmd.h>
#include <X11/Xlib.h>
#include <X11/Xresource.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/keysym.h>
#include <X11/Shell.h>
#include <X11/Xatom.h>
#include <X11/Xaw/Form.h>
#include <X11/Xaw/Label.h>
#include <X11/Xaw/Command.h>
#include <X11/Xaw/Box.h>

void cmdBut1()
{
    printf("Button 1 was pressed\n");
}
void cmdBut2()
{
    printf("Button 2 was pressed\n");
}
main(argc,argv)
    int argc;
    char **argv; {
    XtAppContext Context;
    Widget toplevel,frm1,but1,but2,lb11;
    Window outw;
    toplevel = XtAppInitialize(&Context,"ex1",
        NULL,0,
        &argc, argv, NULL,
        NULL,0);
    frm1 = XtVaCreateManagedWidget("topform", formWidgetClass, toplevel,
        XtNresizable, True, NULL);
    lb11 = XtVaCreateManagedWidget("Hello World", labelWidgetClass, frm1,
        XtNresizable, False,
        XtNfromHoriz, NULL,
        XtNfromVert, NULL,
        XtNleft, XtChainLeft,
        NULL);
    but1 = XtVaCreateManagedWidget("Button1",commandWidgetClass, frm1,
        XtNresizable, False,
        XtNfromHoriz, NULL,
        XtNfromVert, lb11,
        XtNleft, XtChainLeft,
        NULL);
    XtAddCallback(but1, XtNcallback, cmdBut1, NULL);
    but2 = XtVaCreateManagedWidget("Button2", commandWidgetClass, frm1,
        XtNresizable, False,
        XtNfromHoriz, but1,
        XtNfromVert, lb11,
        XtNleft, XtChainLeft,
        NULL);
    XtAddCallback(but2, XtNcallback, cmdBut2, NULL);
    XtRealizeWidget(toplevel);
    XtAppMainLoop(Context);
    exit(1);}
```

When a button is pressed, the widget handles the animation feedback, such as inverting colors or 3D effects.

X Windows handles widgets differently than other windowing systems. In X Windows, the widget library implements the appearance and the feel of the widget (e.g., getting an event to redraw itself).

The Athena Widget library comes with the standard X client library set, and Motif is a commercial implementation that adds a 3D look to your X Windows system.

## PROGRAMMING LANGUAGES

The most common programming language for RTOS applications is probably C, and most GUI programming environments have C libraries.

Widget libraries for X Windows, even though they're object oriented, are written and called by C. Some C++ based application libraries are also compatible.

Listing 1 is a sample C program using the Athena Widget library (Xaw). Xaw doesn't have the fancy look and feel you get with Windows or Motif, but it's smaller.

Listing 1 simply opens a windows, says "Hello World," and creates two buttons. Since widgets are event driven, I assigned call-back functions to the buttons, which are called when a button is pressed.

Once the widgets are programmed, `MainLoop` is called. This code deals with

all likely events (e.g., uncovering Windows-handling mouse-button press events). Photo 1 shows it all.

HTML is making its mark as a GUI programming language, even though it's not a standard programming language like C or Pascal.

HTML-based GUIs rely on HTTP, the protocol for transferring information on the Web. The server (i.e., the application) sends HTML files to the client (i.e., the Web browser), which renders the display and handles

user inputs. The Web server is often embedded in the RTOS application and can be fairly small because all the hard work is done on the node running the browser.

Another popular language for GUI development is Tk/Tcl, which has two components—toolkit (Tk), the GUI's interface to the language, and task-control language (Tcl), the interpreter for the programming language.

Tk is widget oriented and commonly implemented on X Windows, although implementations exist for Windows and MacOS as well. Although Tk/Tcl is inter-

preted, it's widely accepted because it makes GUI development portable.

And because Tk/Tcl is interpreted, it's easy to test new ideas and alter the interface. Tcl is small and written in C and has been ported to many OSs and RTOSs.

And then, there's Java. Java is more than just a language. It's really three components.

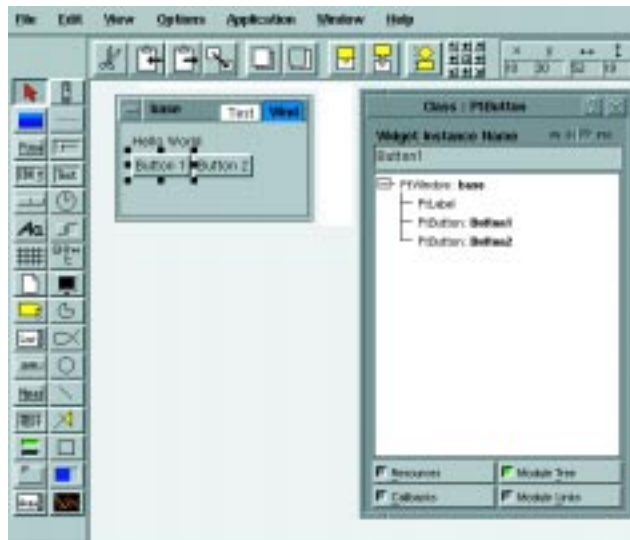
The Java language is a C++ like object-oriented language that promises to be the new standard in programming languages (it's less complex than C++). Although first implementations of Java compilers generated Java byte code for the Java virtual machine, there are now Java-compatible compilers that target native code.

The original intention of the Java system was to use the Java language to compile into Java byte code, which is platform/architecture independent and interpretable by a Java virtual machine (JVM).

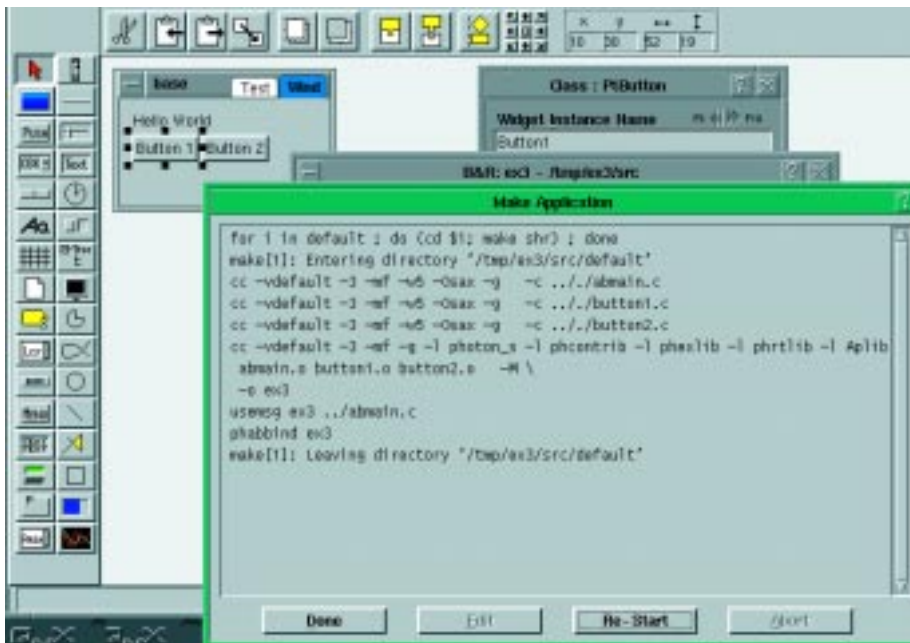
What does this have to do with GUI development? Java also defines a standard library or toolkit for GUI programming, which developers can use to program GUIs to run on any platform JVM and toolkit have been ported to.

Typical implementations of the JVM and toolkit are limited to Web browsers like Netscape Navigator and Microsoft Internet Explorer. However, some OS vendors, such as WindRiver and MicroWare, include JVMs as plug-ins for their OSs.

Because the environment is relatively new, there are still growing pains. For example, Sun, the developers of Java, and Microsoft, one of the licensees of

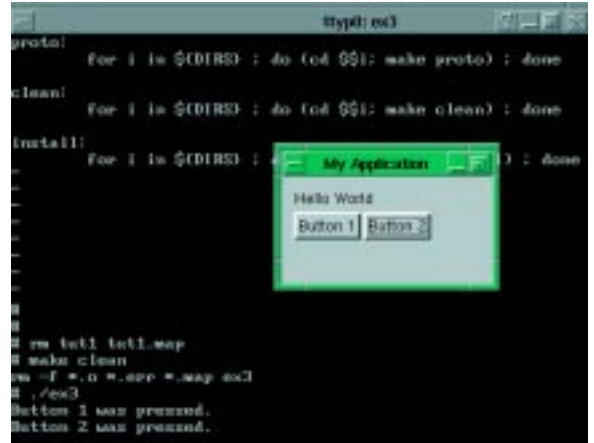


**Photo 2—In this typical setup for building an application with Phab, you can see the GUI prototype as well as how the widgets are wired together in the application.**



**Photo 3—Once the GUI is designed, the C source modules are generated and the project is built using make. The complete development cycle can be handled from Phab, which also functions as a project manager.**

**Photo 4—The GUI has been designed and compiled. Now, the stand-alone program implements the prototype GUI.**



Java, each have their own implementation of Java compilers, virtual machines, and toolkits. These systems have subtle differences, so true cross-platform compatibility is impractical for some applications.

Although embedded Java implementations of the toolkit and JVM are available, they're large (8+ MB image). Embedded

Java and JavaOS are apt to go head to head with WinCE, especially in markets like set-top boxes or smart communication devices.

**Listing 2—The program generated by the application builder looks quite different from Listing 1. Much of the configuration is handled by global data structures, which are implemented in the various header files, also generated by Phab.**

```

#ifdef __USAGE
%C - This is a QNX/Photon application
%C [options]
Options:
  -s server           Server node or device name
  -x x               Initial x position
  -y y               Initial y position
  -h h               Initial h dimension
  -w w               Initial w dimension
Examples:
  %C -s4              Run using Photon server on node 4
  %C -s//4/dev/photon Same as above
  %C -x10 -y10 -h200 -w300 Run at initial position 10,10 with
                       initial dimension of 200 x 300
#endif
#include <stdio.h>      /* Standard headers */
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <Ph.h>         /* Toolkit headers */
#include <Pt.h>
#include <Ap.h>
#include "abimport.h"  /* Local headers */
#include "proto.h"
#include "abwidgets.h"
#include "ablinks.h"
#include "abevents.h"
#include "abvars.h"
char ab_exe_path[PATH_MAX]; /* AppBuilder globals */
void main (int argc, char *argv[])
{
  ApInitialize(argc, argv); /* AppBuilder Initialization */
  ApClassInit(); /* Set up class table used by this application */
  ApLinkWindow( NULL, &appl_links[0], NULL ); /* Display main window */
  PtMainLoop(); /* Loop until user quits application */
}
void ApClassInit()
{
  ApAddClass("PtWindow", &PtWindow);
  ApAddClass("PtLabel", &PtLabel);
  ApAddClass("PtButton", &PtButton);
}

```

JVM relies on garbage collection to manage memory. Current implementations aren't deterministic with respect to garbage collection, so you can't use Java in systems with hard real-time requirements.

But, GUIs only require soft real time. So, Java is a viable option if the JVM is implemented as a separate low-priority task.

To help with Java's dynamic memory requirements, a fixed amount of memory can be allocated to the JVM task. A pool of memory is then preserved for tasks needing traditional memory-management techniques.

## INTERFACE BUILDERS

Interface builders run on the development host and enable you to layout and design the GUI. These tools even have GUIs themselves. So, let's use an interface builder to design that little two-button "Hello World" GUI. I'll use QNX's Phab (Photon Application Builder).

Incidentally, Photon looks a lot like X to the developer. But, its underlying architecture is designed to work with QNX's RTOS in a real-time environment.

Start Phab in Photon's desktop under the developer tab or by typing `phab` into a console window. On startup, Phab displays a standard project-manager-type development interface.

Next, drag widget objects you want into an application window. Several display tools let you view the application architecture. Photo 2 shows a typical session.

Extract it in `Makefile` and various C program modules and header files to implement this GUI, and go ahead and build the application (see Photo 3).

Or, use `make` to compile the project outside the interface builder (see "Software Development for RTOSs," *INK* 93). Once the application is compiled and executed, it should look like Photo 4. Listing 2 shows the Phab-generated code.

Once the GUI is designed, interface it with the application by implementing callbacks for buttons, menus, and so forth.

## LOOK OUT

Now you've seen two implementations of GUIs for embedded systems—X Windows and Photon. But, that's just the beginning.

There are many books on X Windows, Motif, and Tk/Tcl, not to mention all the Java literature. There isn't much on embedded real-time PCs, but you can pretty much

ignore this issue with respect to GUIs because they're soft real-time applications. Just make sure the GUI for your application doesn't affect its hard real-time response.

Well, with a GUI, at least you have something to look at. But I'm guessing you want bells and whistles, too. So, next time, I'll discuss multimedia applications for real-time PCs. [RPC.EPC](#)

*Ingo Cyliax has been writing for INK for two years on topics such as embedded systems, FPGA design, and robotics. He is a research engineer at Derivation Systems Inc., a San Diego-based formal synthesis company, where he works on formal-method design tools for high-assurance systems and develops embedded-system products. Before joining DSI, Ingo worked for over 12 years as a system and research engineer for several universities and as an independent consultant. You may reach him at [cyliax@derivation.com](mailto:cyliax@derivation.com).*

### SOURCES

#### Photon microGUI, Phab

QNX Software Systems Ltd.  
175 Terence Mathews Cres.  
Kanata, ON  
Canada M2M 1W8  
(613) 591-0931  
Fax: (613) 591-3579  
[www.qnx.com](http://www.qnx.com)

#### SVGA cards

Ampro Computers, Inc.  
990 Almanor Ave.  
Sunnyvale, CA 94086  
(408) 522-2100  
Fax: (408) 720-1305  
[www.ampro.com](http://www.ampro.com)

Real Time Devices, USA  
200 Innovation Blvd.  
State College, PA 16804-0906  
(814) 234-8087  
Fax: (814) 234-5218  
[www.rtdusa.com](http://www.rtdusa.com)

Versallogic Corp.  
3888 Stewart Rd.  
Eugene, OR 97402  
(541) 485-8575  
Fax: (541) 485-5712  
[www.versallogic.com](http://www.versallogic.com)

WinSystems, Inc.  
715 Stadium Dr.  
Arlington, TX 76011  
(817) 274-7553  
Fax: (817) 548-1358  
[www.winsystems.com](http://www.winsystems.com)

#### Keyboard interfaces

Vetra Systems Corp.  
275J Marcus Blvd.  
Hauppauge, NY 11787  
(516) 434-3185  
Fax: (516) 434-3516

## IRS

- 410 Very Useful
- 411 Moderately Useful
- 412 Not Useful

# A New View

## Part 1: Virtual Instrumentation

*Got a budget limit on test instruments for those one-of-a-kind gadgets? Well, don't spend your money. Instead, do what Fred does—generate virtual test equipment using National Instruments' LabVIEW.*

Some time ago, I was talking to an engineer at the Kennedy Space Center about the many types of test equipment engineers find necessary for everyday activities. Of course, the conversation turned to who's who in test equipment and who had what.

As it turned out, I'd spent way too much for all my goodies. "You see," the engineer stated, "We don't buy test instruments anymore. We generate them."

Generate them? Hey, if I could "generate" a few Tek scopes from thin air, I'd be rich. Well, once I returned from dreamland, it hit me what he was really saying. His engineering team had converted to virtual instrumentation.

At the Space Center, there's an abundance of one-of-a-kind widgets designed specifically for a single mission to those places "where no one has gone before."

And, every engineer knows that special widgets sometimes require special test equipment.

It could get (and is) really expensive tooling up new test fixtures and specialized test gear for every little quirk project that comes along.

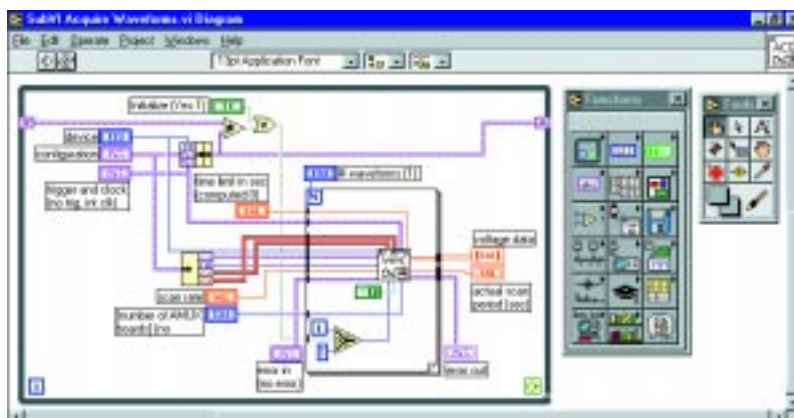
Virtual instruments? What a concept! Sounds like article fodder to me!

### LOOKING AROUND

Virtual instrumentation is one of those phrases that defines itself. Virtual implies that the entity exists but not in a form that can be grasped or physically handled. And instrumentation is, well, instrumentation. The words "virtual instrumentation" lead me to thinking in terms of software emulation of said instrumentation.

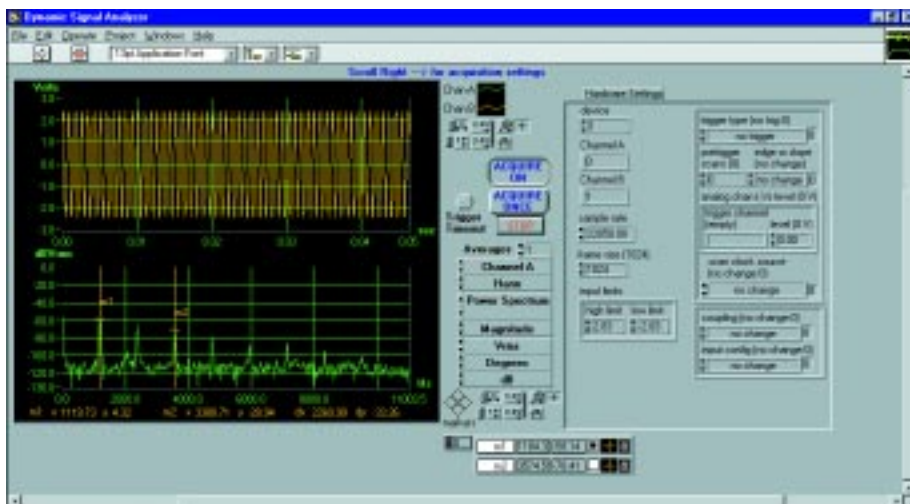
To me, a good definition of virtual instrumentation is a fully functional test platform built with software containing all the physical properties of its hardware counterpart. With software being the working entity of the instrument, the only hardware you'd need to put a virtual instrument to work would be the input conversion circuitry.

Of course, as I was listening to the engineer



**Photo 1—Looks like trouble! Don't worry, it looks complicated, but the process is really very logical.**





**Photo 2—All that trouble in Photo 1 gives way to this human interface.**

describe some of the applications his team used virtual instruments for, I was thinking about how I could employ this technology in the embedded-PC world. If they could run their stuff on a laptop, I could surely run it on an embedded hummer. I walked away from this encounter with bunches of ideas for applying virtual instrumentation to the embedded PC.

After some market research in the back of a bunch of all those free technical magazines I receive, I decided to get familiar with the National Instruments offering. After all, their ad says, "The Software is the Instrument." Can't get any more to the point than that. Well, I picked up the phone and here's what happened.

**AT FIRST SIGHT**

"Re: Hardware, Software, and Manuals. Oh my!" That's how the header of my letter from Erin Nelson at National Instruments read. What an understatement!

The hardware I received consisted of a single PCMCIA (or PC Card for the enlightened of you) that made up the hardware side of a virtual DMM. I also received a ready-to-run DMM VI (virtual instrument) package, which I'll put to use in a later installment.

As for software, how about a full-blown version of National Instruments' LabVIEW V.4.1 for Windows 95/NT. And manuals? There's about 5.5" worth.

The bounty of printed material includes a LabVIEW tutorial, a LabVIEW data-acquisition basics manual, a LabVIEW user manual, a LabVIEW code interface reference manual, and a LabVIEW function and VI reference manual. It took me a

couple of hours to read through the tutorial alone, which brought me to a realization of how I should approach this subject of VI.

There's simply too much information to convey in a single article. For those who are neophytes to LabVIEW and data acquisition, it would be unjust to feed them by firehose. On the other hand, for the old dogs acquiring data in their sleep, it would just be more techno-babble to sort through to the new and innovative ideas.

OK. We're all some kind of engineer or at least an engineer wannabe here, so let's apply some simple logic. Ever read a primer about anything you were new to? Or, did you jump right into the advanced text?

Did you ever laugh at a newbie in your shop who had to learn the ways of a world

you'd been in for years? Remember when you were a newbie?

Repeat after me, "LabVIEW primer."

**HAVE A LOOK-SEE**

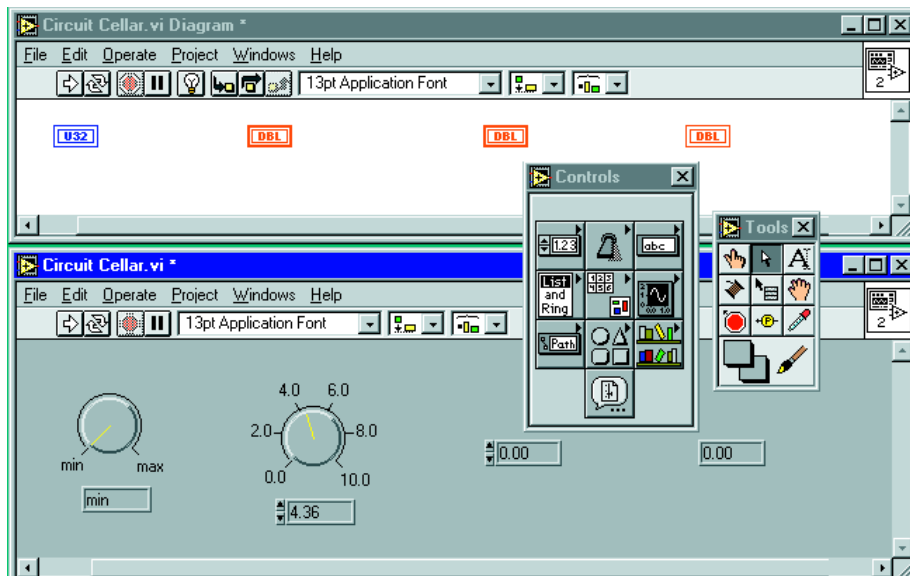
LabVIEW is short for Laboratory Virtual Instrument Engineering Workbench. Remember some time ago, I wrote a piece on machine shop G code? Well, G is back but in a different light. LabVIEW sequences are assembled with what National Instruments calls G language.

G can be thought of as graphical source code. Instead of lines of code, G is "pictures" of code. In other words, LabVIEW's terminology is geared towards the technician, scientist, and engineer.

Graphical symbols are used instead of lines of text as source for LabVIEW applications. You'll see how this works as we get deeper into this project. LabVIEW is also communications ready and can be interfaced to GPIB, VXI, RS-232, RS-485, and plug-in data-acquisition boards.

As you can ascertain from the description, LabVIEW is a flexible development environment composed of built-in libraries and instrument drivers. Standard programming tools like breakpoints and program single-step are also incorporated into LabVIEW.

This flexibility enables custom instrumentation applications to be built easily by both novice and expert data-acquisition engineers.



**Photo 3—As you can see, the Controls Palette is graphically designed to be easy to use and understand.**

### LabVIEW VIEW

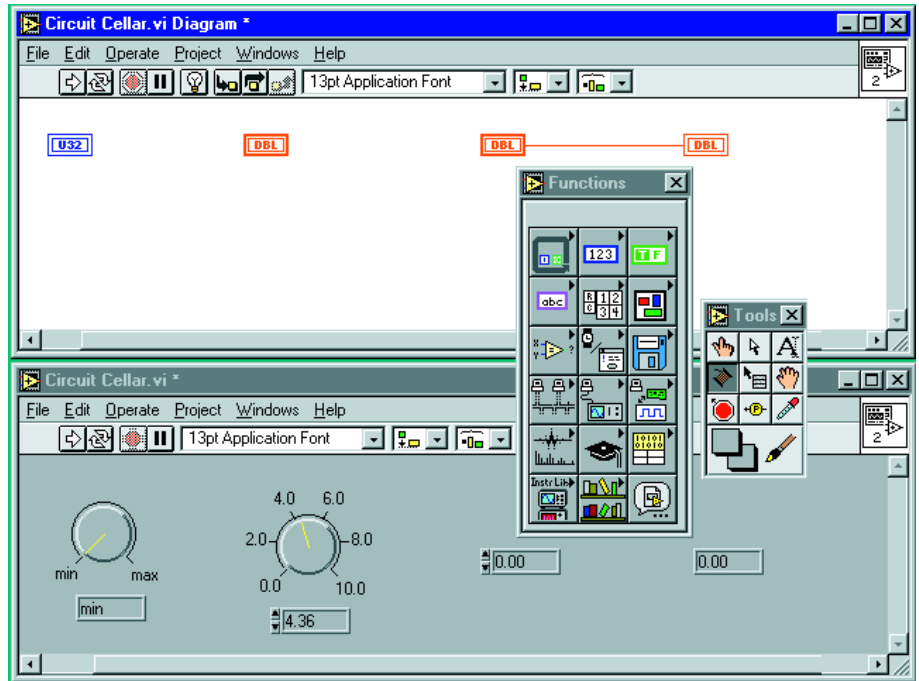
As I stated, LabVIEW is a virtual-instrument environment. Instead of adjusting dials and switches on your physical instrument with your hands, you adjust them on your virtual instrument with a mouse, keyboard, or touchscreen. LabVIEW provides a visual interface that's much like the physical interface of the instrument it's emulating.

LabVIEW applications are created using graphical programming. Nodes are LabVIEW objects that execute functions. These nodes are assembled on the computer screen and resemble a flow chart.

Tying independent nodes together with "wires" enables execution results to flow from node to node. Photo 1 is a typical representation of LabVIEW nodes and wires configured to make a dynamic signal analyzer virtual instrument.

There are two main parts to any VI—a front panel and a block diagram. The front panel is the actual user interface. Photo 2 is the front panel on the other side of the block diagram you see in Photo 1.

The block diagram is the graphical source code. Controls are obtained from



**Photo 4—Note that every control is represented by a corresponding object in the block diagram and the Functions palette is similar in design to the Controls palette.**

the Controls palette. When a control is selected and placed on the front panel, a companion terminal is placed on the block diagram.

There are two types of front panel objects—controls and indicators. Controls are usually data sources, whereas indicators, well, indicate. Indicators look a lot

like controls, but the thicker border around the control sets them apart.

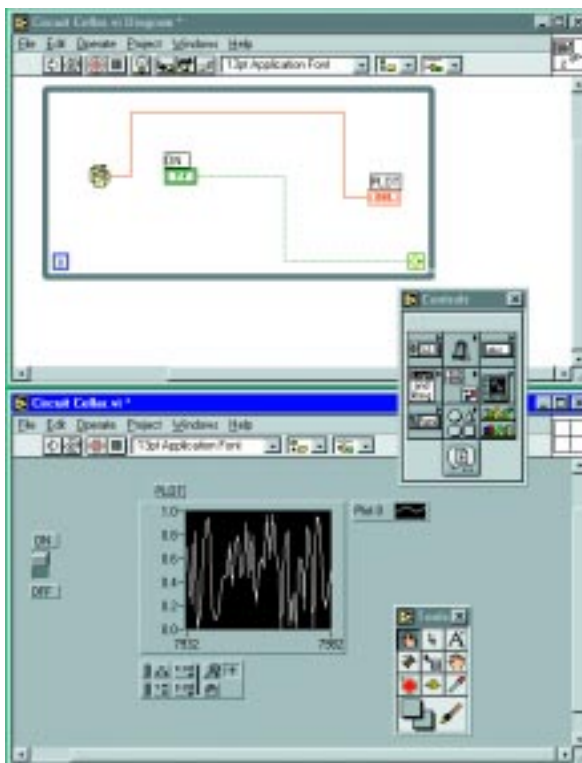
Pop-up menus let you change the characteristics of front-panel objects. Photo 3 is a shot of various controls and indicators along with the Controls palette.

Block-diagram nodes are accessed via the Functions palette. The Functions palette is where the VIs, custom VIs, and functions live. Functions are built into LabVIEW and thus don't have "bodies" like front panels and block diagrams do.

Execution order within a block diagram is governed by data flowing from one node to another. Photo 4 shows us the Functions palette along with objects associated with control and indicators on the front panel.

Functions and subVIs don't work until all of their respective inputs are available. A function in LabVIEW behaves just like a function you write in a common language like C.

Think of a subVI as a textual subroutine equivalent. A VI is really treated just like a textual subroutine in LabVIEW.



**Photo 5—This is absolutely the easiest do-while loop I have ever written (or drawn)!**

and used as a subVI in a block diagram. In other words, VIs used in other VIs are called subVIs. By assembling your LabVIEW task into multiple VIs, you can obtain a top-down programming model just like one you'd implement in a standard text-based programming language like BASIC or C.

SubVIs also contain a third component: the icon/connector. The icon/connector is the interface between VIs used as subVIs. The icon is the visual representation of a VI when it is used as a subVI. SubVIs at the block-diagram level normally include an input and output terminal. This is the way subVIs connect logically to each other.

The output terminal of a downstream subVI connects to the next logical upstream subVI input terminal. The location of the input and output terminals are defined by the connector. Both the

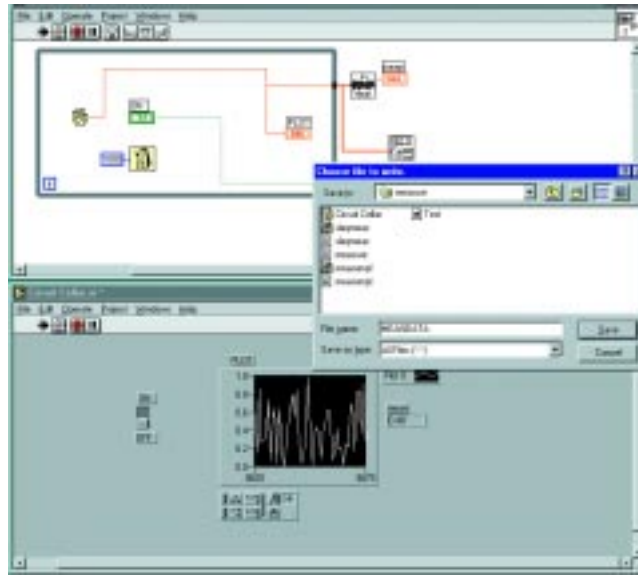
Normally, VI functions should be kept as simple as possible. Once a particular VI is realized, it can be reduced to an icon

icon and connector can be edited from the front panel. A connector pane represents each front-panel object.

For instance, if a front panel consisted of one digital control and one digital indicator, the connector pane would consist of two terminal panels. One panel would represent the control output terminal and the other panel would designate the indicator input terminal. Control terminal panes are usually on the left side of the connector pane and indicator panes are on the right.

In that LabVIEW can output data to a spreadsheet file, execution order can sometimes be very important. In other cases that don't require external file I/O operations or statistical number crunching, the order of execution can be relatively lax.

Wires connect nodes on the block diagram. Because there are various types



**Photo 6—It's not as complex as the instrument in Photo 1, but this one is just as functional.**

of data flowing between nodes, it would seem that there should be a scheme to differentiate what data is flowing over what wire. LabVIEW delineates data types by color. For instance, green represents Boolean values, while orange objects and wires denote floating-point values.

## LOOKING SHARP

Yep, pretty neat stuff, huh? It doesn't take long to get up to speed with LabVIEW. If you've ever written a single line of code, this logical development environment will come to you quickly. Let's walk through building a simple virtual instrument.

Selecting the New VI from the LabVIEW menu screen results in screens we saw in the earlier photos sans controls and indicators. The first thing we want to do is assemble a front panel. Assume we want to graph some random data. We'll choose a switch and a graphical window for the front-panel objects. To keep it simple for now, that's all we'll put on the front panel.

The next step is to put the remaining block-diagram elements in place. We already have a couple of block-diagram objects (terminals) that correspond with the front-panel objects I just added.

Since I don't have a real signal to track, I'll emulate an input signal by applying random numbers into the VI's input. This is done easily in LabVIEW by selecting the Random Number Generator VI from the Functions palette. This VI generates a random number between 0 and 1.

Earlier I mentioned that LabVIEW's programming techniques paralleled standard text-based techniques. Here I'll use the graphical equivalent of a do-while loop structure to control the logic of our simple VI. As you can see in Photo 5, the while loop is actually a boundary that runs all of the code contained within its boundaries while the conditional terminal is TRUE. I'll tie the front-panel switch which is a Boolean to the conditional terminal.

The final task needed to complete our simple VI is to logically attach all of the terminals' inputs and outputs. In our case, that involves connecting the Boolean switch to the while loop's conditional terminal and connecting the output of the Random Number Generator terminal to the input of the graphic display window. This is really a "door bell" LabVIEW app, but this is the primer. As you can imagine (because I can't make the graph move on this page), the resulting waveform is displayed on the graphic window in Photo 6 when the Boolean switch is turned on or TRUE.

Most data-acquisition schemes don't move this quickly. There are limits to available acquisition memory and in most cases available power over time. So, it would be nice to have a means of selecting time intervals for certain data gathering applications.

LabVIEW handles this with the Wait Until Next ms Multiple VI. To incorporate this feature into our VI, I simply select the icon from the Features palette and plop it onto the block diagram inside the while-loop boundary. Once the timer icon is in place, a right click of the mouse button enables me to create a constant. The constant contains the number of milliseconds to delay.

In today's world, spreadsheets have moved off the banker's desk onto the engineer's workstation, and it's not enough to just collect the data. These days, everything's analyzed. Well, to be politically correct, LabVIEW can do spreadsheets and analysis.

In our application, we can calculate the mean of our random readings by placing the Mean VI outside our existing while loop and connecting the output of the RandomNumber VI to the X input. At the same time, spreadsheets are covered by placing the Write To Spreadsheet File VI in the same vicinity.

Our data is one-dimensional, so the output from the Random Number VI will connect to the 1D input of the spreadsheet VI. This action forms a small black tunnel on the while loop boundary. This tunnel is the logical exit for the data being fed to the Mean and spreadsheet VIs.

To pass the collected data as a data set to the Mean VI, a right mouse click on the tunnel lets me enable indexing. Indexing allows the while loop to collect the random-number data and pass it to the Mean VI when the loop terminates.

To see the mean value on the front panel, I need to create an indicator. This is done by right clicking the Mean VI and choosing the Create Indicator option. Once that's done, the mean indicator is placed on the front panel. Photo 7 brings all of this to light.

## PEEKING AHEAD

There's only about five more inches of doc left, but I'm out of time. The intent here was to introduce you to LabVIEW, while at the same time planting the embedded-PC seed. Now that the basics have been covered, I can move on to mating this

virtual world to the physical world we live in every day.

Next time, I'll install LabVIEW on an embedded platform and gather real data. Once you see how easy it is to create embedded virtual instruments, you'll see I've once again proven that it doesn't have to be complicated to be embedded. [APC/EPC](#)

*Fred Eady has over 20 years' experience as a systems engineer. He has worked with computers and communication systems large and small, simple and complex. His forte is embedded-systems de-*

*sign and communications. Fred may be reached at [fred@edtp.com](mailto:fred@edtp.com).*

### SOURCE LabVIEW

National Instruments Corp.  
6504 Bridge Point Pkwy.  
Austin, TX 78730  
(512) 794-0100  
Fax: (512) 794-8411  
[www.natinst.com](http://www.natinst.com)

### IRS

413 Very Useful  
414 Moderately Useful  
415 Not Useful

# FEATURE ARTICLE

Don Lancaster

## PostScript Flutterwumpers

### Pick a Peck of PostScript PICs

The sound of “flutterwumpers” brings on a sort of Alice in Wonderland enchantment that’s not unlike the peculiar magic Don Lancaster pulls out of PostScript. Listen in on how to make PostScript move PIC-powered robots.



I’m a long-time fan of the superb, general-purpose PostScript language. For years, it’s been my first and foremost tool for most of my columns, engineering designs, and technical illustrations.

I also use PostScript for many other tasks, ranging from investment models to magic sine-wave research, Smith charts, shaft encoders, Web-site traffic analyzers, and hot-tub controllers. And as you’ll see, it also lets us explore some low-end robotics opportunities.

Until recently, you had to want to use PostScript as a language in the worst sort of way to be able to do so. In particular, needing a PostScript printer to serve as the host computer was horribly limiting. PostScript hard drives were few and far between, as was the lack of a real-time visual display. Even more crippling was the one-way parallel port that nearly all early PC systems forced on PostScript users.

But, these limitations are ancient history. PostScript is now busting out all over with brand-new capabilities. For example, the free Ghostscript 5.01 clone provides full host-based PostScript computing combined with full-screen visuals.

Variations on Ghostscript can be customized for new commercial apps. In fact, Videonics now offers the PowerScript 1000 fully animated video character generator, thus turning zillions of fonts and all the standard desktop-publishing tools loose on video editing.

Originally, PostScript lacked most transparency features. But, Videonics’ workaround redefines the CMYK color space, with K serving as a new alpha transparency channel.

Host-to-printer interactions have dramatically improved as well, with higher speed two-way parallel communications added to serial, AppleTalk, and Ethernet options. Program apps such as Download Mechanic simplify PostScript-as-a-language and printer interactions.

Meanwhile, Adobe improved its Acrobat 3.01 PostScript variant, too. Acrobat greatly revolutionizes the distribution of information, both off and on line.

I find it much better than HTML because it offers exact control of what the end user will see, with full-screen magnifiable displays online and improved antialiased visibility. Acrobat also features URL hot linking, interactive forms, wipes, full text and library searches, bookmarks, thumbnails, byte range loading, scan capture, and powerful handicapped-access provisions.

A typical single-file size is 11 KB per illustrated page, which prints equally well to PostScript and non-PostScript printers. And one element of Acrobat, the Distiller 3.01, lets you convert complex PostScript code into simpler Acrobat .PDF files.

#### ACROBAT DISTILLER

A PC-, Unix-, or Macintosh-based Acrobat Distiller 3.01 makes a fairly powerful general-purpose PostScript computer. At its input, Distiller accepts PostScript code from a disk text file. It can also be taught to read nearly any file format in any language!

At the output, Distiller can generate its (usually intended) .PDF file or write messages or calculated results to its (usually ignored) .LOG file. Alternatively, it can write any custom-disk-based text file in almost any format.

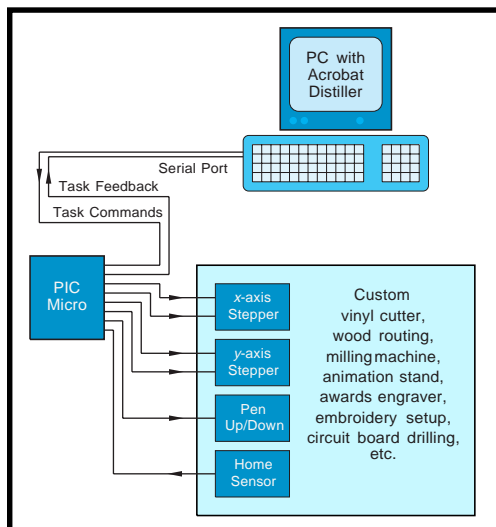


Figure 1—Flutfile partitioning brings genuine PostScript to PIC flutterwumpers.

Distiller can easily analyze Website log files or extract special content from private search-engine internals. It can also convert file and font formats, as well as filter all the Fourier transforms from millions of magic sine waves, keeping only those scant few good ones. It can even design digital filters or solve higher order linear equations.

The final results can be displayed onscreen using Acrobat Exchange, Reader, or Ghostscript. Both Exchange and Reader let you print to either PostScript and non-PostScript printers.

Let's go over a simple example that shows how fast and easy it is to use PostScript as a computing language—finding the sine of 35.4°. First, create this text file:

```
%!
% Find sine of 35.4 degrees
35.4 sin ==
```

If you drag and drop this file into Distiller, the value of 0.579281 should pop up on your log-file display.

Distiller obeys the rule that any print (e.g., print or ==) commands go to the .LOG file. Write commands (e.g., write or writestring) are output to your custom .TXT disk output file. Marking commands (e.g., show, stroke, or showpage) go to the generated .PDF file.

The log file is intended for error and status messages, but you can write nearly anything to it at any time for any reason. Unfortunately, the log file isn't quite transparent. It has a few

hidden restrictions, so you have to decide whether to pick the quick and dirty "free" log file or write a custom output file.

Let me go over some of the log-file limitations. Adobe Distiller does a flushfile after every new line, which is nice because you can see problems as they occur. However, it slows you down and causes a lot of disk clatter on extended outputs.

Also, long strings are truncated to 200 characters in the log file. Carriage returns are tricky as well, and you get a mix of status, error, and data messages.

One big gotcha when writing any PostScript-as-language code that needs PC disk access: always use "\\\" when you mean "\" inside a PostScript string. The confusion comes about because "\" acts as a directory boundary in Windows filenames, whereas it's a fully reserved PostScript string character. Ghostscript gets around this by substituting "/" for "\" in filenames.

And finally, don't be alarmed if you see "Warning: No PDF file produced." If your primary goal is writing your own custom disk file, you may not need a .PDF viewable file at all.

## DISTILLER VS. GHOSTSCRIPT

Distiller is amazingly fast and 100% genuine Adobe. Ghostscript, by contrast, is slow, klutzy, and not at all friendly. Ghostscript clearly flaunts its Unix heritage, but a new View add-on provides a graphical interface.

Both Distiller and Ghostscript offer visual displays. But even with the two mouse clicks required to switch to Exchange, Distiller displays are faster and considerably clearer.

Ghostscript lacks some of the latest PostScript features, and it has some unofficial bugs. However, it can be customized to make anything you like. Ghostscript font additions are a hassle, but Ghostscript source code is easily obtained. In short, Distiller costs money; Ghostscript doesn't.

Ghostscript offers one debugging advantage. Distiller gags on any error, outputting only a stack dump. Ghostscript gives you a visual screen output up to the point your error was made, thus showing you where to pin down an obtuse bug.

Nevertheless, I prefer Distiller, perhaps because I'm an Adobe developer and beta tester. The use of Distiller for PostScript-as-language that I am the most excited about involves...

## PostScript FLUTTERWUMPERS

A "flutterwumper" is my term for any autonomous process controller whether it be hardware or software based. Some examples include printed circuit drillers, wood routers, fabric shears, panel engravers, animation stands, cutters for vinyl, embroidery machines, shop mills, laser trimmers, and box makers.

As Figure 1 illustrates, PostScript provides a unique way of partitioning any flutterwumper, greatly reducing the cost of dedicated hardware.

The key is to let PostScript handle the higher level stuff—managing disks and other resources, interfacing design software or clip-art libraries, dealing with fonts, doing coordinate transformations, making tool-path corrections, or converting curves to individual machine space steps. After the hard stuff is done, your onboard PIC only needs to deal with the simplest tasks.

To start this process, use transfer documents, which I call flutfiles. Two obvious languages for flutfile documents are Gerber Format and HPGL. Note

0	Go one step due east
1	Go one step northeast
2	Go one step due north
3	Go one step northwest
4	Go one step due west
5	Go one step southwest
6	Go one step due south
7	Go one step southeast
B	Begin sequence
D	Pen down or cutter on
H	Reset to home position
Q	Quit sequence
Rn	Repeat last n - 32 times
U	Pen up or cutter off
X	Breaking debugger
%	Ignore comments until CR

Table 1—Here's one possible flutfile command set. The single characters handle elemental tasks.





Let's take a look at three simple PostScript flutfile examples. Since flutfiles are all resolution dependent, let's assume that the required system resolution is 100 steps per inch, or 10 mils per step.

Figure 2 shows a simple rectangle. First, you home with pen or cutter up. Then, shift to the lower left corner of the rectangle and drop the pen. Next, trace the entire rectangular path, raise the pen, and return home.

To get from PostScript into your flutfile, you need a PostScript routine that intercepts PostScript commands, finds their paths, converts the paths to straight-line vectors, resolves these vectors down into machine-dependent steps, and then creates a flutfile. These tasks are all trivial in PostScript.

A PostScript routine builds a flutfile by intercepting the `moveto`, `lineto`, `curveto`, and `closepath` operators. Since nearly anything PostScript can be redefined at any time for any purpose, all these operators are redirected to generate the output flutfile commands. Acrobat automatically reduces complex PostScript code into these fundamental elements.

The usual way to deal with fancy artwork is to send it on through the Distiller first and then print to disk to extract the elemental PostScript commands. Those commands are then pasted into a PS flutfile generator. Or, they are read and interpreted directly from a disk textfile.

Flutfiles can be sent in real time or saved to disk and reused for production. You can also host-create elemental flutfile libraries. Several ready-to-use examples of suitable tools are in `FLUTOOLS.PS` on the Flutterwumper Library Shelf at `<www.tinaja.com>`.

Note that any existing PostScript code application can generate all of your desired paths. Although I prefer to write custom code in raw PostScript, using Illustrator or any PostScript-compatible drawing program works just fine. In fact, nearly any existing PostScript or Acrobat program can automatically generate robotic flutfiles.

This means your robotic system will automatically speak PostScript from day one—royalty free and using nothing but a PIC or two. Figure 2 shows us the PostScript code for a simple rectangle.

A version using repeats and removal of unneeded characters is also shown, which dramatically shortens this particular flutfile. Any positioning commands now take slightly longer to execute because they're limited to 0-7 cardinal moves. If you always use inches as the unit of measurement, you can simply do a `72 dup scale` at the start of your file instead.

Figure 3 shows a 3" circle. As you can see, the code is even simpler. Figure 4 depicts a Palatino R. Again, you can use any PostScript artwork creator or these simple raw PostScript commands.

One minor gotcha: The font you pick must already be ATM resident in your host. If it isn't, Courier is substituted for misspellings or non-Adobe fonts. A rectangular bounding box is

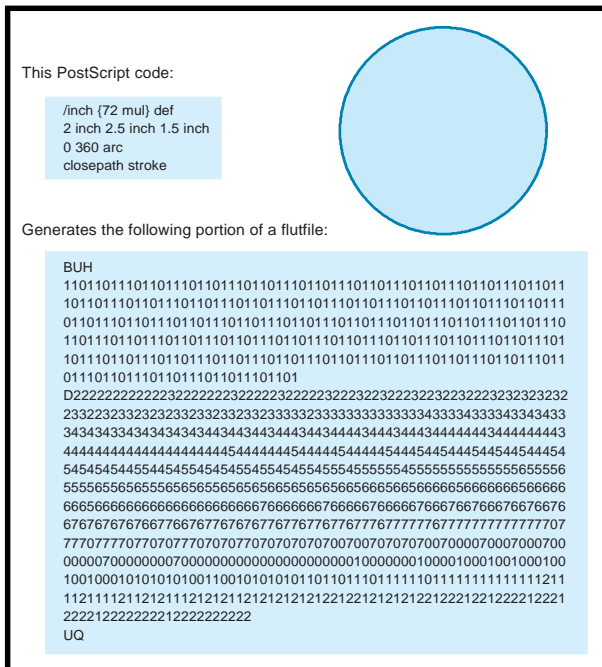


Figure 3—This code generates a large 100-dpi circle.

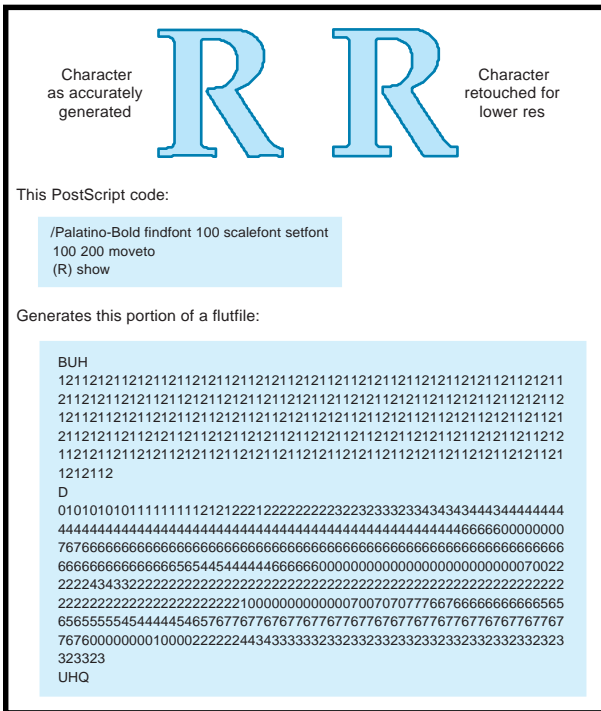


Figure 4—Flutfiles easily handle shapes as detailed as a Palatino R.

substituted for any real but unavailable Adobe fonts.

A second minor point: Notepad has problems with new line control characters, so use Wordpad or some other editor instead.

Above all, it's important to realize that your flutterwumper electronics don't care about all this fancy stuff. All your custom hardware has to do is convert ASCII characters into simple steps, which is something a PIC can do in its sleep.

## IMPROVING RESOLUTION

Flutterwumper resolution tends to be much poorer than what you see in any laser printer. Obviously, there isn't much point in routing out some large wooden display sign closer than, say, 1/16". And, it is real tricky to force better than 10-mils resolution from any larger positioning hardware.

A flutfile can be set to almost any resolution, so there's no limit on how fine you slice things. But on coarser systems, there are a few tricks you can pull that greatly improves your results. With practice, you can easily "sight read" any flutfile and make adjustments on-the-fly.

First, retouch any grungy flutfile elements so they look best at your chosen resolution. For example, the

initial Palatino R in Figure 4 looks a little jaggy and unbalanced. The retouched version seems somewhat less "Palatino-ish" but more suited to a lower resolution flutterwumper.

Retouching is slightly different than the usual font hinting processes for coarse bitmaps because diagonal paths are permitted. It's especially useful for character or libraries of forms or other locations where standardized code is reused a lot.

A second resolution-enhancing trick is step-per-phase interleaving. Most steppers employ four phases or incre-

ments to complete a step. Each phase moves you one quarter of the way towards your goal.

So, as Figure 5 shows, instead of jumping a whole x then y step on a 1-3-5-7 slant move, you do a quarter x then quarter y step. Your paths end up much smoother and less jaggy.

For low-cost larger systems, you can convert any automotive alternator into a cheap power-stepper motor. Alternator conversions often use three, not four, phase steps, but the concept of stepper phase interleaving remains the same.

## PostScript P.S.

So, now you have some powerful new tools to dramatically lower the cost, reduce the complexity, and im-

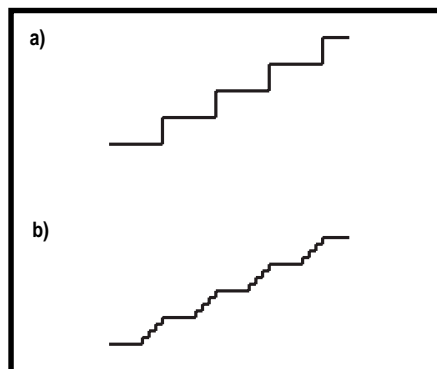


Figure 5a—Normal vector-to-step conversion produces a quite jagged output. b—By contrast, stepper phase interleaving improves the apparent resolution.

prove the performance of any low-end to midrange robotics you develop.

Just split your motion-control problem in thirds. You only need to use the simplest dedicated PIC hardware on your robot. Have this hardware respond to a flutfile or similar "elemental motions only" intermediate file structure you create in PostScript by using host-based Acrobat Distiller or Ghostscript application utilities. ☐

*Don Lancaster is the author of 35 books and countless articles. Don maintains a U.S. technical help line at (520) 428-4073 and also offers books, reprints, and consulting services. You may reach him at don@tinaja.com.*

## SOFTWARE

To learn more about using PostScript as a computing language, check the tutorial `DISTLANG.HTML` at <[www.tinaja.com/acrob01.html](http://www.tinaja.com/acrob01.html)>. Flutfile generators and utilities are at <[www.tinaja.com/flut01.html](http://www.tinaja.com/flut01.html)>. Information on Ghostscript is available at <[www.cs.wisc.edu/~ghost](http://www.cs.wisc.edu/~ghost)>.

## SOURCES

### Acrobat, Exchange, Distiller

Adobe Systems, Inc.  
345 Park Ave.  
San Jose, CA 95110-2704  
(408) 536-6000  
Fax: (408) 537-6000  
[www.adobe.com](http://www.adobe.com)

### Download Mechanic

Acquired Knowledge, Inc.  
3655 Nobel Dr., Ste. 380  
San Diego, CA 92122  
(619) 587-4668  
Fax: (619) 587-4669  
[www.acquiredknowledge.com](http://www.acquiredknowledge.com)

### PowerScript

Videonics, Inc.  
1370 Dell Ave.  
Campbell, CA 95008  
(408) 866-8300  
Fax: (408) 866-4859  
[www.videonics.com](http://www.videonics.com)

## I R S

416 Very Useful  
417 Moderately Useful  
418 Not Useful

## DEPARTMENTS

66

MicroSeries

72

From the Bench

78

Silicon Update

# EMI Gone Technical

## MICRO SERIES

Joe DiBartolomeo

## Transient-Suppression Design Technique

Part  
4  
of  
4

To tie together all his ideas on protecting against EMI and transient threats, Joe presents the protocol he uses when designing for transient protection. These general guidelines help you stop EMI from sneaking through.



In the previous installments of this MicroSeries, I've given you a look at the most common types of transient electromagnetic threats and their associated waveforms.

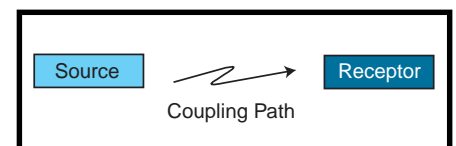
You've also become well-acquainted with the components most commonly used to protect electronic equipment from these transient threats.

In this final article, I take all the discussion and put it together as a straightforward design protocol. It's the method I use when designing transient protection.

### DEFINING THE PROBLEM

When you're designing transient protection—or any EMI protection for that matter—keep in mind the classic source-path-receptor model shown in Figure 1.

For an EMI problem to occur, you need three components—a source of EMI, a receptor susceptible to it, and a coupling path between the source and receptor. Although this model is normally applied on a systems level, it is also valid on a circuit level.



**Figure 1**—The source-path-receptor model is often used as a starting point for detailed EMI designs and solutions.

The solution to any EMI problem lies in breaking this chain. You can eliminate the source of EMI, break the path between the source and receptor, or as I'll discuss in this article, design the receptor so that it's not susceptible to transient EMI.

At the systems level, the source of the transient and the coupling path are rarely under the control of the receptor designer. So, to protect against transient threats at the system level, you have to protect the receptor.

However, the situation improves when the transient enters or is generated within the equipment. Inside the equipment, you have some control over the coupling path and, in some cases, the transient source.

Another thing to keep in mind when designing for transient protection is that transients are RF problems. This factor complicates the process.

In RF design, things aren't what they seem to be. A wire is a not a wire—it's a resistor and an inductor. Components such as capacitors vary greatly with frequency, depending on lead inductance and composition.

For an example of how components change at high frequencies, take a look at the diode in Figure 2a. The ideal diode blocks the negative portion of the sine wave and passes the positive portion.

However, many diodes exhibit this ideal behavior only at low frequencies. As the frequency of the input sine wave increases, the diode junction capacitance comes into play, as illustrated in Figure 2b.

As a result, at high enough frequencies, the diode passes both the positive and negative portions of the AC waveform. This nonideal behavior can fool you into thinking you have protection when you don't.

In Figure 2c, an electrostatic discharge (ESD) is applied to a reverse-biased diode. With a risetime of 0.7 ns, the ESD has frequency components into the gigahertz. Therefore, the ESD is coupled across the diode via junction capacitance.

You can think of the diode junction capacitance as a hidden element. At higher frequencies, all components have them. These hidden elements

can make it difficult to design transient protection. It's a good idea to draw out your circuits with all the hidden elements, thereby exposing any hidden paths a transient might take.

But please, don't get intimidated by RF and EMI and think you need to solve Maxwell's equations. Start with the basics and go from there.

For example, plan the route the transient current will take. Don't leave it to chance. Transient currents are normally routed to ground, so make sure you have a good RF connection to a good RF ground.

Place suppression components as close to the devices being protected as possible. Keep leads to suppression components as short as possible. Basically, follow good RF design practices.

Now, there are many approaches to hardening a system or circuit to transients—probably as many approaches as there are designers. Of course, I have my own, which I consider fairly simple. In fact, my approach is more a general thought process than a technique. I missed my calling as a philosopher.

The best way to illustrate the process is by example. Let's say you're designing transient protection for a piece of test equipment—an automatic circuit-board tester or oscilloscope.

These types of instruments are normally connected to the mains (lightning). Their attached I/O cables are subjected to ESD and electrical fast transients/bursts (EFT/B).

## PROTECTION ZONES

My first task is to divide the equipment into three zones or levels of required protection—interface, inter-board, and IC—as illustrated in Figure 3. Each zone has its own characteristics and requirements.

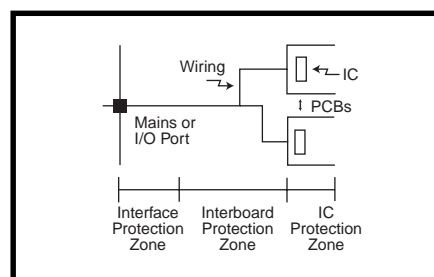


Figure 3—Between the three zones—interface, inter-board, and IC—an incoming transient is suppressed, with each zone absorbing its portion of the transient.

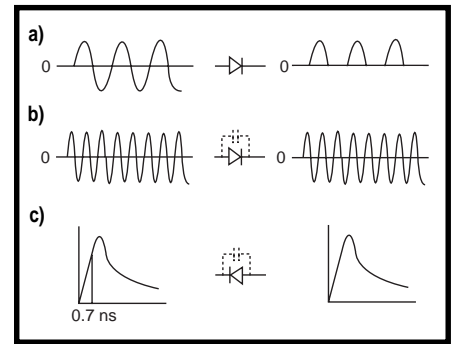


Figure 2a—The ideal diode blocks the negative portion, whereas the nonideal diode (b) passes the entire sine wave due to junction capacitance. c—The very high-frequency components will get coupled across the diode even though it's reverse biased.

When considering the interface-protection zone, I am concerned with how as well as how much of the transient will enter the equipment.

At the inter-board-protection level, I'm thinking about what happens once transients enter the equipment and are distributed via the wiring to the PC boards.

And at the IC level, I'm considering what happens once a transient reaches the individual IC.

The idea of the three protection zones is quite simple. Among the three, any transient will be suppressed because each zone serves a particular purpose. That is, it handles the portion of the transient it's responsible for and leaves the remainder for the other zones.

In general, the components used in each zone have specific characteristics. These are listed in Table 1.

The components in the interface-protection zone can handle the greatest amount of transient energy, thereby preventing as much of the transient as possible from entering the equipment. They also tend to have slower turn-on times and are coarse with high turn-on voltages.

One example of a component used in the interface-protection zone is the gas discharge tube. Gas discharge tubes can handle a great deal of energy, but their turn-on voltages are far too high to protect individual ICs.

Even though interface-protection components are what transients normally encounter first, it is the inter-board- or even the IC-level protection components that turn on first. There-

fore, another important function of the interface-protection component is to protect the interboard and chip-level transient suppressors.

The interboard-protection components bridge the gap between the interface and IC-level components. These devices have two primary functions—to turn on before the interface components and to protect the IC-level components.

By comparison, the chip-level protection components, which protect the individual ICs, have the fastest turn-on times and lowest on-voltages but they handle the least energy. Depending on the transient, IC-level components are the first to turn on.

But as I mentioned, IC-level suppressors depend on the other suppression components for protection. On its own, a TVS diode provides little protection from a lightning transient.

In Part 3, I presented a table of specification for each type of transient suppressor, and in Part 1, I gave the technical details of the transient waveforms. Taken together, they should make it fairly straightforward for you to select suppression components for the interface, interboard, and chip-level protection zones.

Any component may be required in any protection zone. For example, at the interface-protection zone, there are basically two places a transient can enter the system—the power mains and the I/O ports.

The transients that the mains sees normally differ from those encountered at the I/O ports. The mains are subject to lightning and EFT/B, whereas ESD normally isn't a concern. I/O ports on the other hand, are subject to

Zone	Current Capabilities	Turn On	On Voltage
Interface	High	Slow	High
Interboard	Moderate	Moderate	Moderate
Chip	Low	Fast	Low

**Table 1**—Suppression components used in each zone have general characteristics.

ESD, but lightning and EFT/B may not be of concern, depending on the application.

However, take care that the zone can handle the current generated by the transient. For example, the wiring to a gas tube must be able to handle the currents generated by lightning.

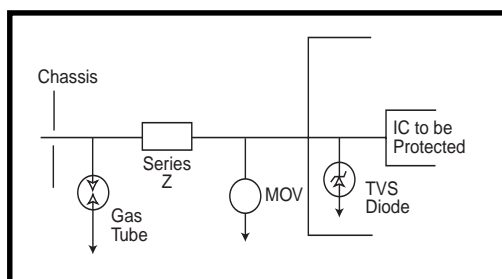
If a gas tube is placed on a PC board, then the tracks of the PC board must be able to handle the lightning's transient current.

## SUSCEPTIBLE POINTS

After breaking the system down into three zones, I identify points within each zone where transients can enter. I also identify every point in the system that is susceptible to transients.

I then draw the circuits, as the transient would see them—from every susceptible point to every entry point. For each schematic, I ask several questions:

- what are the transients I expect to see at this point, and what do their waveforms look like?
- what level of protection is required?
- what protection components are at my disposal?
- how will the protection component protect my circuit/system?
- how will the protection components affect the operation of my circuit/system?



**Figure 4**—In this generic protection scheme for an I/O line, each suppression component handles a specific function. Together, they provide complete protection.

So, let's put all this information together and see how it works.

Figure 4 depicts a generic line being protected. At the interface to the outside world is a gas tube. A series impedance leads to the metal oxide varistor (MOV) in the interboard zone, and the IC-protection zone has a TVS diode.

Generally, there are two types of transients—those that

are very fast with low energy content (e.g., ESD) and those that are slow but with high energy content (e.g., lightning). What happens when either one of these transients hits the system?

In the case of ESD, the interface components are of no use. The transient will have come and gone by the time the protection device turns on.

Recall that a gas tube takes 100 ns to turn on, and the ESD pulse is typically in the nanosecond range with a rise time of less than 1 ns. An MOV turns on in about 50 ns, so it too is unable to follow the ESD.

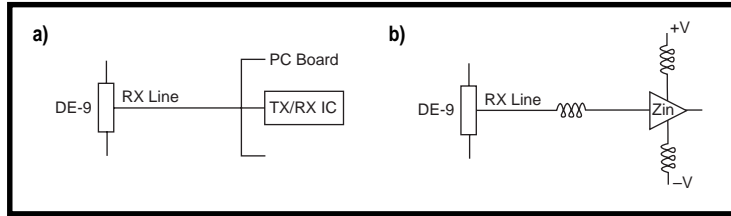
Faced with an ESD, only the board TVS diodes can provide protection. Luckily, the TVS diode has enough energy-handling capability to suppress the ESD on its own.

In the case of lightning, the rising edge generally gets by the gas tube devices. This is known as overshoot.

The MOV can normally follow the leading edge of the lightning, and if anything gets through, there's still the IC-level protection. However, since MOVs derate with use, it's important that the series impedance and gas tube protect the MOV after it has handled the lightning's leading edge.

Figure 5 gives another example. Here you see a typical RX line on an RS-232 link. The RX line comes in via a DE-9 connector and is wired to the TX/RX IC. How do you protect the RX line on the transceiver IC?

Draw the schematic at the RX input to the transceiver. What does the input to the transceiver look like, and what is connected to the transceiver? (Don't forget the power pins on the IC.)



**Figure 5a**—Here's a standard RX line interface. **b**—The addition of the hidden components make it much easier to design for transient suppression. You should repeat this technique at every transient-susceptible point.

Identify transient inputs that can couple to this point, and draw out the coupling circuit for each transient source. Then ask yourself the above questions.

The RX line could be exposed to any or all of the common transients—ESD, EFT/B and lightning. However, ESD is most likely the biggest threat. Clearly, the power lines and the RX line to the DE-9 will be sources of transients. Depending on your system, however, more transient threats could arise.

Standards require RS-232 lines to be able to withstand up to 15 kV of ESD. There are TX/RX ICs that can handle transients of this size without needing external suppression components. However, the level of protection required may be more than 15 kV.

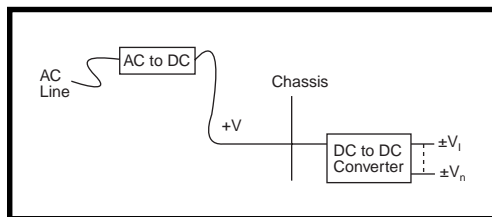
Since the RX line is most likely to see ESD, the protection components are limited to the TVS and perhaps zener diodes. If it were my choice, I'd go with TVS diodes (see Part 3).

Place the TVS or zener diode in shunt with the RX line, and protect the RX line by diverting the transient current to ground. The capacitance of the TVS diode or the zener won't affect the edges of the RS-232 line, but for higher speed links, you should consider the shunt capacitance of the protection devices.

## GET THE DETAILS

Detailing every point in this manner may seem a little obsessive. Surely, some points can be ignored. However, even when it seems that transient protection is well under control, detailing it points out potential problem spots.

Figure 6 illustrates a simple power supply I designed using two commercial bricks. The external brick converts the AC to DC, and an internal brick provides DC-DC conversion.



**Figure 6**—To avoid EMI problems, designers may buy off-the-shelf power converters. However, you need to ensure that the EMI specs of the commercial converters meet the EMI specs of the final system.

With this setup, I avoided a lot of the transient-protection details. The designers of the bricks took care of it, right? Oops, this could get me into a lot of trouble.

There's no doubt the designers of the bricks provided transient protection,

but just because the transient protection was sufficient for their needs doesn't mean it's sufficient for mine. I need to reask myself the questions I asked above.

Normally, ESD isn't a power-port problem because the mains are wired into the unit. ESD is most common when a person comes in contact with the system/circuit. However, with the external brick, ESD becomes a problem that I'll have to address.

Well, there you have it—my approach to transient design. As you see, I take a general approach filling in the details on a case-by-case basis. Why? Because the interaction between a transient and a system or circuit depends as much on the system or circuit as it does on the transient.

## HEADS UP

On a related note, a new version of the generic immunity standard for residential, commercial, and light industry is now in place. EN 50082-1:1997 replaced EN 50082-1:1992 and came into effect March 1, 1998.

If you ship to the European Community, be aware that the new standard expands the immunity tests. For more information on EMI tests, please refer to my MicroSeries in *INK* 79-83. 📧

*Joe DiBartolomeo, P. Eng., has over 15 years' engineering experience. He currently works for Sensors and Software and also runs his own consulting company, Northern Engineering Associates. You may reach Joe at [jdb.nea@sympatico.ca](mailto:jdb.nea@sympatico.ca) or by telephone at (905) 624-8909.*

## IRS

419 Very Useful  
420 Moderately Useful  
421 Not Useful

# Rebirth of the Z8

## FROM THE BENCH

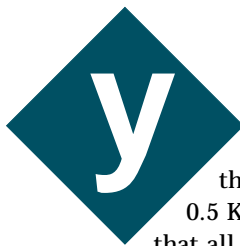
Jeff Bachiochi

### Part 2: Let Your Micro Answer All



Want to see the future? Jeff's project

merges a piece of the past (remember your old Magic 8-ball?) with Zilog's new OTP processors. It's crystal clear how easy it is to implement one of these new Z8s.



You can't do anything useful in 0.5 KB. I used to hear that all the time.

Nowadays, we know better. There are plenty of small tasks out there that can be performed adequately by the new generation of small processors.

Last month, I talked a bit about Zilog's new OTP line of processors using the proven Z8 core. This time, I want to base a project on Zilog's bottom-of-the-line device, the Z86C02.

This 18-pin DIP or SOIC device has 11 I/Os plus three inputs, which can be either digital/interrupt or comparator inputs. The single timer can be externally clocked or gated. What could I possibly do with this little beastie?

And then it rolled right in front of me. What's black and round, looks like an enlarged billiard ball, and answers all the great questions of the young? I had one as a child. My youngest now plays with one.

And it looks like the Magic 8-ball has survived another fall to the ground.

For those of you who aren't familiar with the 8-ball, it is a liquid-filled sphere that has a small clear window. You ask the 8-ball a yes/no question as the ball is agitated.

One side of a polygon within the liquid then settles against the clear window, revealing the 8-ball's remarkable prophecy. What better introductory project for this Zilog processor than the Mystic Z8-ball shown in Photo 1?

This project requires an LCD, a couple of switches, and a few batteries. The smallest single-line LCD I could find was an eight-character display. That's too small because I want the whole message to be displayed at once.

I settled on a  $1 \times 16$  character display. I also wanted to have at least 16 separate messages. So let's see, 16 messages times 16 characters, that's 256 bytes of text. Wait, that's half the available code space! This doesn't look good.

#### HARDWARE

Despite the queasy feeling in my stomach, I went ahead and drew up a schematic, shown in Figure 1. Notice that SW1 applies the power to the circuitry. This microswitch is located just inside the Z8-ball's base. It closes when the ball is lifted from the desk, coffee table, or other flat surface.

The second switch, SW2, is a mercury switch that opens when the LCD (mounted in the base) is tilted up and closes when it's flipped back over (or agitated).

Penlight (AA size) batteries power



**Photo 1**—I used a plastic ceiling globe for the Z8-ball. It's slightly larger than the original 8-ball. The circuitry is barely larger than the  $1 \times$  LCD module.

the processor LCD and backlighting for the display. Although the display's backlight is by far the highest current draw, the on-time is characteristically short, so the batteries should last a long time.

Most parallel LCDs can be set up to run with either an 8- or 4-bit data path. Having plenty of I/O pins available, I opted for the simpler 8-bit mode.

Handshaking with the LCD can be accomplished via the RD/\*WR control line. The LCD signals that it's busy processing data by holding the busy data bit high until it's again ready for more data.

But, this particular feature can't be used during the reset sequence. The reset sequence is regulated by strict minimum data-to-data timing. Since the ability to handle this timing must be incorporated for the reset sequence, you can use this minimum timing as a guide to the readiness of the LCD without getting into a more complicated test for busy/not busy.

Very few different LCD controller chips are used on LCDs. And, the ones that are used are so flexible they can interface with many glass formats, alone (for small LCDs) or expanded with additional segment drivers.

But, this flexibility doesn't come for free. The cost is an initialization that must be performed to configure the processor for the format of the display it will be driving.

Most controllers adhere to a de facto standard. The controller must know the data path width (8- or 4-bit), the format of the character generator (5 × 7 or 9 × 11), and the number of lines to the display (1 or 2).

Next, the controller needs to know if you want it to shift the characters (scrolling) or the print position (swiping). Finally, you need to enable and disable the display as well as indicate whether or not the cursor should be invisible, blinking, or steady.

Other commands sent to the processor clear the display and enable the cursor to be positioned anywhere on the display. All these commands are

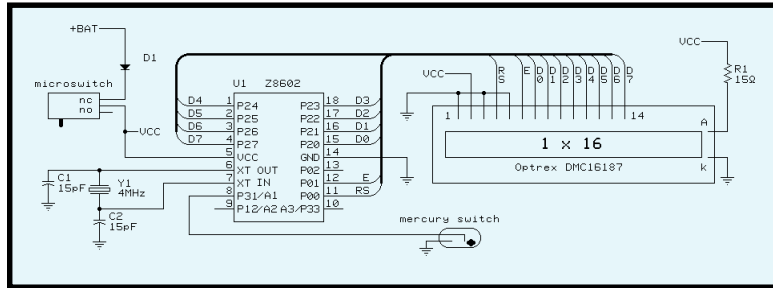


Figure 1—Zilog's OTP processor needs little external support for this simple application.

carried out by communicating with the control register.

To print a character to the display, you need to communicate with the data register. The character passed to the data register is used as an offset into the character generator's table.

The table stores the 5 × 7 character matrix. This matrix is the row/column pixel array which, when transferred to the display position, is a picture representing the character being sent.

## SOFTWARE

I always like to map out my strategy with some simple flowcharts. Figure 2 shows the direction I expected to take. A study of the reset conditions of the Z86C02 processor identifies the registers that need to be initialized prior to getting any real work done.

The code I started out with used only direct register addressing—that is, calling out the full 8-bit address of every register I used. No shortcuts on this first pass.

After initializing all the registers (and I/O ports), I turn my attention to the LCD initialization. The 8-bit LCD mode lets me communicate twice as fast as with the 4-bit mode, but speed here isn't an issue. Only five control bytes need to be passed to the LCD processor to complete its initialization.

There are eight services you can instruct the LCD processor to do. These are discerned by the number of zero bits preceding the first one bit in the control word sent to the LCD processor.

Not all of these services need to be used. At a minimum, it's necessary to set up

on/off).

Since I wasn't going to do any software-busy checking, I needed to pay attention to time. All commands are finished in less than 20 ms. Most take less than 50 μs.

I chose 2 ms as my standard waiting period (clear screen takes 1.64 ms). So, I can clear and print a message in under 50 ms. The wait routine is done as software loops instead of using the timer.

Even though it might be easier (and a lot more accurate) to use the timer, I may not have the code space available to set up the timer for each individual time spec. Besides, this way I can let the timer free run to give me a pseudo-random number whenever the user moves the Z8-ball and closes SW2 without having to set it up to free run each time I need a random message.

Once the user asks a question of the Mystic Z8-ball and turns up the LCD, SW2 opens and the timer is read. The lower four bits of the timer

the 001xxxxx function set, the 0001xxxx shift set, 00001xxx display set, 000001xxx entry set, and 00000001 clear set (see Table 1). The x's are bit selections for choosing the set-up parameters for that service (e.g., cursor

Function Set = 001dnfxx
where d = data width 8/4 bits
n = 2/1 display lines
f = 5 × 10/5 × 7 character font
x = no parameters
Shift Set = 0001wdxx
where w = display/cursor shift
d = shift left/right
x = no parameters
Display Set = 00001dcb
where d = enable/disable display
c = enable/disable cursor
b = blinking/steady cursor
Entry Set = 000001is
where i = increment/decrement position
s = shift/no shift display window with cursor
Clear Set = 00000001
(no parameters)

Table 1—The position of the leftmost 1 determines the control register written to within the LCD.



are masked (number of possible messages to choose from) and the result is shifted (multiplied by 16, the number of possible characters in a display message) to give an offset to the beginning of the chosen (random) message.

The actual message table address is then added to the offset so the block move loop can find the message and pass each character to the display. I chose not to use the canned multiply routine, although Zilog has a number of useful canned routines available.

After displaying the message on the LCD, the program jumps back in anticipation of another earth-shattering question. Turning over or agitating the Z8-ball closes SW2, blanking the display, and the process begins again.

### MASM

With the moment of truth at hand, I load my code into Zilog's MASM, pop a Coke, and watch for the inevitable error messages to appear. Sure enough, they scroll off the screen and end with the message "too many errors."

OK, it couldn't identify some of the register names I used. A little more research corrected most of these errors.

Another pass through MASM and a bunch more errors with the final message "not enough space in ROM for code." Ouch. A view of the .LST file showed the code filling the 0.5 KB of space about halfway into the message table. Hmm....

The I/O port registers I needed to write to were in the first working register bank (the first group of 16 registers), so I mapped the registers I needed into that space. The Z8 has 16 groups of working registers, with the control registers residing in the upper bank.

By changing from direct register addressing to indirect register addressing, I could save a byte on each of those instructions. By setting the working register group number in the RP register, I can call out registers of interest using a nibble (indirectly) instead of a byte (directly). This improvement made a big dent in the code space needed, but it wasn't enough.

A look at each of the routines showed the LCD initialization requiring

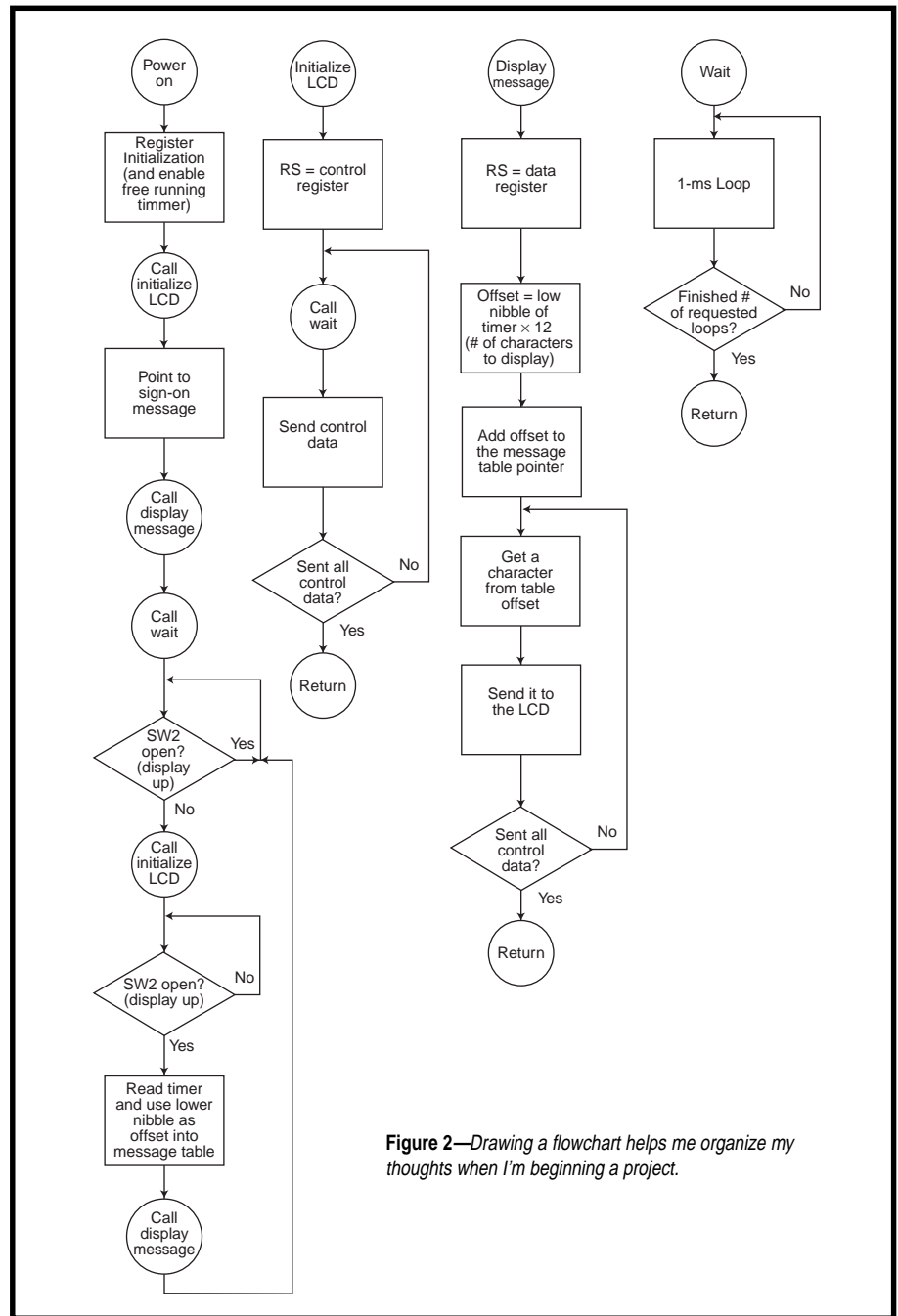


Figure 2—Drawing a flowchart helps me organize my thoughts when I'm beginning a project.

a big chunk of code. Each control word written had a lot in common.

Given the millisecond-loop setup, I could move the data strobing into the millisecond loop where it would take up minimum space. True, that's not where it belongs, but I could still jump into the middle of that wait routine and avoid strobes when I was timing things other than the period between LCD writes.

Now, I was able to assemble the code and stay within the bounds of the Z86C02's available memory. However, a successfully assembled

file does not necessarily make a logically correct process.

### Z8 ICEBOX

I don't want to guess how much this ICEBox costs Zilog to produce, but for \$99, it's a real deal. Getting these things into engineer's hands is bound to help Zilog get designed in, and that's really the bottom line.

There are a number of jumpers on the Z86CCP00ZEM in-circuit emulator. It certainly pays to read the documentation before trying to use this board.

The 'ZEM has typical windows software for keeping track of general-purpose and control registers, timers, I/O, and the like. The debug window permits single-stepping, step-over, and breakpoint entries. You can run your code in real-time or animation (automatic single stepping) mode.

I found myself missing the resettable execution counter found on other tools. This counter enables you to take accurate timings of specific routines.

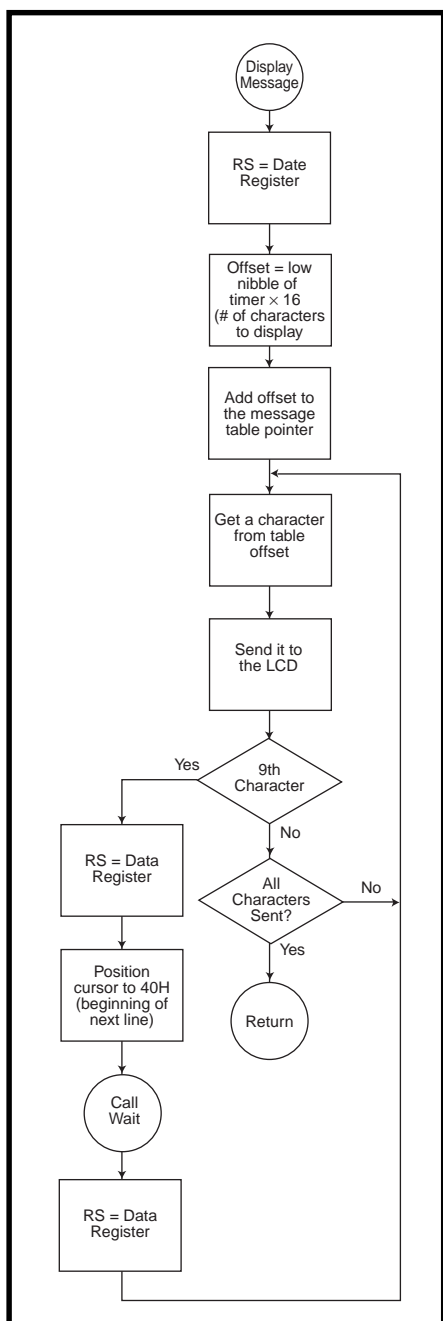


Figure 3—Adjustments to the display routine were required because the 1 × 16 LCD turned out to be a 2 × 8 display.

In place of a cycle counter, I often flip an unused output bit going into and out of the routine in question. With a scope, it's easy to monitor the bit and get an accurate reading of the routine's timing.

After setting up the 'ZEM, attaching my circuit with the included 18-pin cable, and downloading the assembled code, I clicked Go. The display jumped to life by printing eight 5 × 7 blocks.

Well, these blocks are typical for a display that hasn't been initialized. So, not only was the initialization code not working, but only the first half of the display was working.

A look at the LCD's datasheet revealed the words "16 character × 1 line." I was setting it up right, right?

I seem to remember that, many times, the display isn't exactly as it seems. The 20-character × 4-line displays I used were in fact 40 characters × 2 lines. The last half of each line was chopped off and placed under the first two lines.

So, if you didn't control the positioning, the message started out on the first line, continued on the third line, jumped up to the second line, and continued on the fourth line. Could the datasheet be misleading?

A look at the LCD's block diagram indicated that the processor could handle two 64-byte segments. Could this display be 8 character × 2 lines?

I had to append the `display message` (see Figure 3) routine to include a test for the ninth character. When the ninth character was encountered, the cursor had to be repositioned to the 40H byte before the final eight characters were displayed.

Unfortunately, this added enough source code to put me back over the top again. Arrgh.

A second look at the timing routine simplified the code enough for an assembly within bounds (two bytes shy of full). By using a single-byte loop register as the minimum time increment (750 μs), I could use a byte pair (word) as an outer loop counter (that's up to 50 s).

The `DECW` instruction lets the register pair be decremented (it takes care of the byte boundary carry task), saving a few instructions over the

three-byte, three-nested loop routine I used previously. Success at last.

I'm nearly done. As I look over the display routine, I see that I set up the routine to use any number of random choices. It so happens that I'm limited in this program to 16 choices.

Instead of multiplying my lower timer nibble (1 of 16) by 16 (the number of characters in a message), I could have just used the `SWAP` (nibbles) instruction and saved a bunch of rotate instructions. Oh, well. There's always room for improvement!

## JUST ASK!

Presentation is a big part of this project, and finding the right spherical container can be a challenge. For the best effect, try to make it resemble the original 8-ball as closely as possible. Then, you're ready.

Pick up the Z8-ball and turn it over, and the backlit LCD reads "Mystic Z8-ball." After a few seconds, another message appears, "Ask and shake me."

Asking the mighty know-it-all, "Do you truly have psychic powers?" I shake it wildly. Without a second thought, the Z8-ball answers, "No, I doubt it."

Ahh. I guess this seeing into the future doesn't work after all. But wait a sec...if the Mystic Z8-ball knows it doesn't have psychic powers, doesn't that mean it does? ☹

*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@circuitcellar.com](mailto:jeff.bachiochi@circuitcellar.com).*

## SOURCE

### Z8, ICEBox

Zilog, Inc.  
210 E. Hacienda Ave.  
Campbell, CA 95008-6600  
(408) 370-8000  
Fax: (408) 370-8056  
[www.zilog.com](http://www.zilog.com)

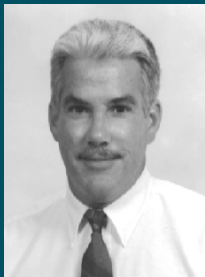
## IRS

422 Very Useful  
423 Moderately Useful  
424 Not Useful

# SILICON UPDATE

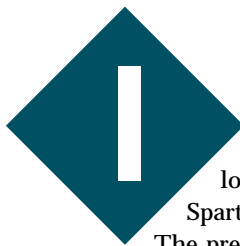
Tom Cantrell

## FPGA Tool Time



It's fine to have a low-cost FPGA, but if the

tools cost a mint, you're still in trouble. However, Tom's found some bargain basement tools just as far away as your local Internet connection.



Last month, we looked at Xilinx's Spartan line of FPGAs.

The premise behind their (and competitors like Altera, Actel, and Lucent) bargain-priced chips is simple. If you build it (and the price is right), they (high-volume designs) will come.

Sounds good, but hold your horses. There's the small matter of tools needed to turn a blank FPGA into something other than a high-tech paperweight. In fact, the performance-at-any-price image of FPGAs is as much a by-product of expensive, complex tools as the cost of the chips themselves.

Historically, FPGA suppliers have seemed more interested in making money on the tools than promoting chip design-ins. For many years, Xilinx kept tight control over their place and route technology, giving new meaning to the concept of one-stop shopping. Without access to internal P&R technology, a would-be FPGA tool supplier faces a situation akin to writing a C compiler for a chip with a secret instruction set.

Fortunately, FPGA suppliers have realized that

you'll probably sell more blades if you don't charge \$10,000 for the razor. A bit of browsing in the tool department finds wider selection, more functionality, and lower prices.

### DIALING FOR DEVELOPMENT SYSTEMS

Phil Freidin, local FPGA guru, turned me on to a great programmable-logic Web site. Poke around [www.optimagic.com/lowcost.html](http://www.optimagic.com/lowcost.html), and you'll find your way to a bunch of good deals.

For instance, the Foundation toolset from Xilinx is a shrink-wrap total solution that runs on PCs or workstations and includes everything you need to enter, simulate, and implement a design. However, at \$3995 (\$5995 with VHDL synthesis), it doesn't exactly encourage casual experimentation.

But look closely. You'll find they're offering a promotional version for \$95 that's fully functional but restricted to 8k gates. Better yet, head on over to Associated Professional Systems ([www.associatedpro.com](http://www.associatedpro.com)) and you find the Xilinx promo package bundled with PC-plug in evaluation boards starting at \$340.

Bleeding-edge ASIC designers increasingly use FPGAs to prototype their gate arrays before signing that big, noncancelable PO. Synopsis leverages the trend with FPGA Compiler, a version of their popular Design Compiler ASIC synthesis tools that run under Unix.

They also offer FPGA Express, specifically targeting FPGAs and running on PCs (assuming your box is up to snuff; 64-MB RAM is recommended). A free evaluation version (1k-gate limit) of FPGA Express can be had at [www.synopsys.com/products/fpga\\_pc.html](http://www.synopsys.com/products/fpga_pc.html).

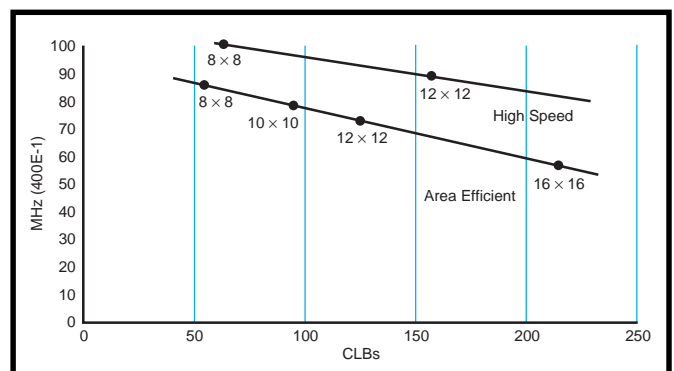


Figure 1—A virtue of FPGA signal processing is that it allows designers to tune price and performance tradeoffs optimally for their particular application.

Exemplar, now a subsidiary of EDA Mentor Graphics, is known as one of the early pioneers of FPGA-optimized synthesis. Their Galileo tool automatically infers technology-specific RAMs, eliminating the need to manually force-fit a particular FPGA's scheme and maximizing device portability.

Similarly, their module generation approach (MODGEN) automatically detects many common arithmetic and relational operators and synthesizes technology-specific solutions rather than random logic. An entry-level package, GalileoFSxi, specifically targets Xilinx FPGAs and includes VHDL or Verilog tutorials. Check out demo deals at [www.exemplar.com](http://www.exemplar.com).

Although it's a relative newcomer, Synplicity is garnering a reputation for quality output and fast compile times with their Synplify package. If your PC is a bit pokey (Synplify is one of the few packages that runs on Windows 3.1 as well as 95, NT, and workstations) and you get tired of hour-long coffee breaks, check out their 20-day evaluation package at [www.synplify.com/downloads/platform.html](http://www.synplify.com/downloads/platform.html).

Other big players and good deals include OrCAD with their Express NT-based VHDL package (get a free demo CD via [www.orcad.com](http://www.orcad.com)), Accolade's PeakVHDL and PeakFPGA ([www.accolade.com](http://www.accolade.com)), and Minc (recently acquired Synario from Data I/O) with their \$495 VHDL Easy ([www.minc.com/html/vhdl0.html](http://www.minc.com/html/vhdl0.html)).

## SYNTHESIS GETS REAL?

Hard to believe it was way back in 1990 when I wrote my first article on the subject of logic synthesis ("VHDL—The End Of Hardware," *INK* 17). Guess I've been having fun because the time's sure flown.

The article introduced VHDL, an Ada-like language originally designed for simulation but called into duty for chip design (like that other popular HDL, the C-like Verilog).

Back then, I predicted that typing in a chip using HDL would replace the traditional schematic

**Photo 1**—Configurable library modules, like this Xilinx filter, insulate designers from the gory details of FPGA-specific architecture and optimization.



approach to hardware development. The problem: schematics would eventually run out of gas, with designers buried under thousands of pages.

Was I right? HDL techniques are now widely used for complex chips. However, the life of schematics has been extended by the development of macros that bury many gates into a single symbol, and they remain many designers' first choice for lower gate-count designs.

Though describing hardware in HDL is much like programming, there are fundamental differences. For instance, software types generally needn't be concerned about the difference between statements like:

$X = A + B + C + D$

and

$X = (A+B) + (C+D)$

However, these statements certainly don't generate equivalent circuits in HDL. The former cascades three adders in series, whereas the latter pairs two adders feeding a third, so it's faster.

Watch out for inferred memory—latches created by `if` statements without an `else` clause (a subject I discussed in "Pick a Peck of PLDs,"

*INK* 68). For instance, confronted with the seemingly innocuous:

```
if (GATE = '1') then
    Q <- DATA;
```

the synthesizer creates a latch to remember the previous value of DATA to know what to output on Q when GATE = '0'.

While a bit of headscratching is the price to pay for explicit control over the hardware implementation, other disturbing gotchas hide in the gap between tool and chip. It's a two-edged sword, imposed by the inability of synthesis tools to both insulate designers from, and grant access to, the specifics of the underlying FPGA architecture.

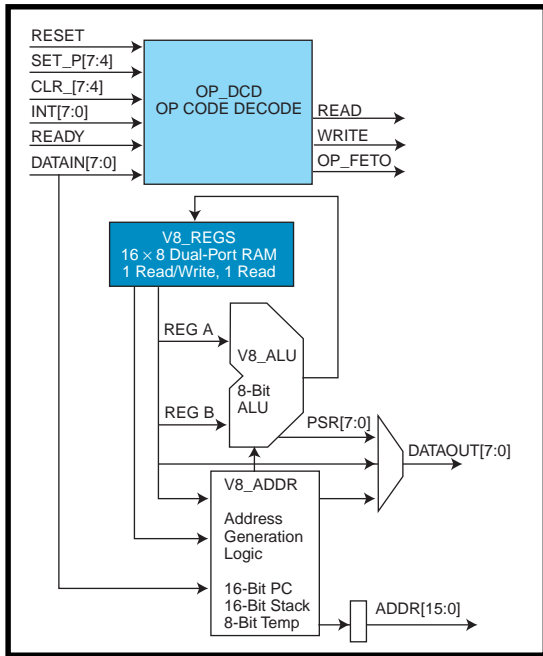
As an example of the former, the obvious way to code a 16-bit barrel shifter is as sixteen 16:1 multiplexers as shown in Listing 1a. Of course, you could get tricky and come up with a

two-tier scheme (see Listing 1b) comprising a 0-, 1-, 2-, or 3-bit rotate followed by a 0-, 4-, 8-, or 12-bit rotate, but that seems dumb.

Or is it? It turns out that for Xilinx chips the latter design is a better fit. It consumes less than half the CLBs (32 vs. 80) and even runs faster (3 CLB delays vs. 4) than the "obvious" design option.

Core	Xilinx CLBs	Supplier
8051, 6502, Z80-compatible CPU	500+ (est.)	VAutomation, Virtual IP Group, Mentor Graphics
V8 8-bit CPU with 64-KB addr. space	268	VAutomation
TX400-series CPU (scalable)	120+	T7L
16450-type UART	130	Comit Systems
Multimaster I <sup>2</sup> C	137	Memec Design Services
HDLC controller	200	CoreEI Microsystems
146818-type real-time clock	100 (est.)	Virtual IP Group
PCMCIA interface library	20-30	Digital Objects Corp.
USB function, hub controllers	600-960	Mentor Graphics

**Table 1**—Mining IP gold starts with a core sampling. The emergence and growth of third-party IP providers enables FPGA system-on-a-chip aspirations.



**Figure 2**—The V8 from VAutomation is the first in a wave of new CPU cores designed specifically for the IP market.

On the other side of the coin, synthesizers' high-level pretensions often mean they're oblivious to nitty-gritty details. For instance, to a synthesizer, signals like clock and reset are no

different than any others. But, as we saw last month, modern FPGAs do understand the difference, offering dedicated pins, routing channels, buffers, and so on.

Treat clocks and reset like any other signal in your HDL code, and you'll end up with some wimpy connection routed hither and yon, leading to all kinds of headaches. Yes, it's possible to get around the problem with nonstandard, device-specific hacks in your HDL, but you'll have to keep track of two versions of your code—one that synthesizes and one that simulates.

The myriad details are hard to learn, and they vary across different vendors' architectures. However, for big chips, there's really no other choice. Echoing Winston Churchill's comments about capitalism, synthesis may be a terrible system, but it's the best there is.

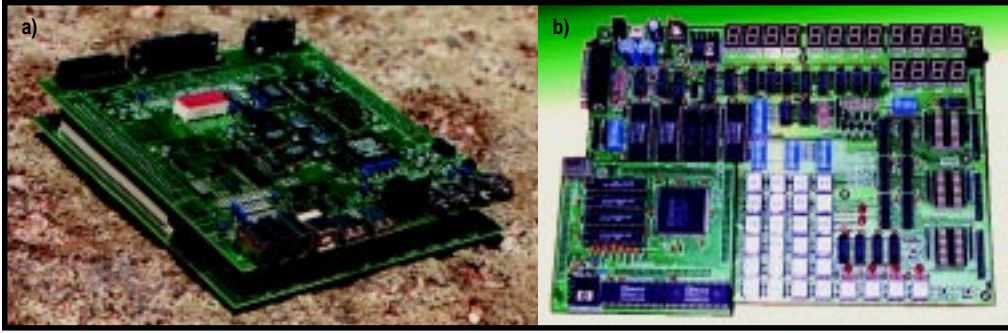
## JUST DO IP

Until synthesizers get to the point where they understand what should be done, rather than just doing whatever you tell them to do, a solution comes in the form of IP (Intellectual Property). It comes from the vendor and encapsulates device-specific know-how.

Don't write your own barrel-shifter. Just pop in a canned one that's already been tested and optimized. Xilinx offers a spectrum of IPs under the umbrella of CORE Solutions, which is composed of reference designs, LogiBLOX, LogiCORE, and AllianceCORE.

Reference designs comprise a collection of application notes describing techniques and/or specific designs for commonly used functions like the aforementioned barrel shifter, counters, FIFOs, PWMs, and so on. Note that these, like typical application notes, aren't warranted or supported. But the price—free—is certainly right.

LogiBLOX goes one step further, wrapping generic functions in a GUI that allows easy modification of parameters such as counter, shifter, decoder,



**Photo 2**—Whether silicon or IP, a CPU needs good support tools including evaluation boards such as these from VAutomation (a) and T7L (b).

or multiplexer width, as well as memory type (ROM, RAM, dual-port RAM) and size. So, rather than having to juggle a library of, for instance, 6-, 7-, and 8-bit counters, the one you need is always just a few point and clicks away.

LogiBLOX also bridges the gap between schematic and HDL design styles. Module lookup and configuration can be invoked directly from third-party schematic editors (e.g., Aldec, Viewlogic, Mentor Graphics, Cadence, etc.), creating a placeable symbol. Alternatively, modules can be instantiated into an HDL design for simulation (both behavioral and gate level) and synthesis.

While LogiBLOX is included with basic development-tool packages, LogiCOREs are higher level, extra-cost functions sold and supported directly by Xilinx. For example, a PCI core enables designers to integrate PC plug-and-play with application-specific logic.

Deliverables include schematics, netlist, and both VHDL and Verilog simulation models. Options such as initiator/target, burst transfer timing, and FIFO size are easily customized to tune performance from 20 to 100 MBps.

The other major LogiCORE offering is the DSP Core Generator, a library of parameterized DSP functions. Like LogiBLOX, the DSP functions (adders,

accumulators, and even complete filters) are configured with point and click dialogs as you see in Photo 1. They also deliver output for both

schematic and HDL design regimes.

DSP applications are an especially good match with Xilinx FPGAs, thanks to distributed-arithmetic algorithms. For instance, a filter distributes coefficients across multiple CLBs, using the LUTs as “ROM.” Sample bits are processed serially, but all taps of the filter are processed in parallel.

Furthermore, unlike a “have it our way” standard DSP, the precision of an FPGA-DSP can be tuned to exactly what’s required. Need a 19-bit multiplier instead of the usual 16 or 24 bits? No problem. In fact, a myriad of choices are available, everything from  $6 \times 6$  to  $32 \times 16$  in both speed- and area-opti-

**Listing 1**—Be careful what you wish for. The straightforward implementation of a 16:1 mux in VHDL (a) consumes more silicon and runs slower than a less obvious version (b) that better matches the FPGA architecture.

```

a) library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_arith.all;
   entity barrel_org is
     port (S: in STD_LOGIC_VECTOR (3 downto 0);
           A_P: in STD_LOGIC_VECTOR (15 downto 0);
           B_P: out STD_LOGIC_VECTOR (15 downto 0);
   end barrel_org;
   architecture RTL of barrel_org is
   begin
     SHIFT: process (S, A_P)
     begin
       case S is
         when "0000" =>
           B_P <= A_P;
         when "0001" =>
           B_P(14 downto 0) <= A_P(15 downto 1);
           B_P(15) <= A_P(0);
         when "0010" =>
           B_P(13 downto 0) <= A_P(15 downto 1);
           B_P(15 downto 14) <= A_P(1 downto 0);
         when "0011" =>
           B_P(12 downto 0) <= A_P(15 downto 3);
           B_P(15 downto 13) <= A_P(2 downto 0);
         when "0100" =>
           B_P(11 downto 0) <= A_P(15 downto 4);
           B_P(15 downto 12) <= A_P(3 downto 0);
         when "0101" =>
           B_P(10 downto 0) <= A_P(15 downto 5);
           B_P(15 downto 11) <= A_P(4 downto 0);
         when "0110" =>
           B_P(9 downto 0) <= A_P(15 downto 6);
           B_P(15 downto 10) <= A_P(5 downto 0);
         when "0111" =>
           B_P(8 downto 0) <= A_P(15 downto 7);
           B_P(15 downto 9) <= A_P(6 downto 0);
         when "1000" =>
           B_P(7 downto 0) <= A_P(15 downto 8);
           B_P(15 downto 8) <= A_P(7 downto 0);
         when "1001" =>
           B_P(6 downto 0) <= A_P(15 downto 9);
           B_P(15 downto 7) <= A_P(8 downto 0);
         when "1010" =>
           B_P(5 downto 0) <= A_P(15 downto 10);
           B_P(15 downto 6) <= A_P(9 downto 0);
         when "1011" =>
           B_P(4 downto 0) <= A_P(15 downto 11);
           B_P(15 downto 5) <= A_P(10 downto 0);
         when "1100" =>
           B_P(3 downto 0) <= A_P(15 downto 12);
           B_P(15 downto 4) <= A_P(11 downto 0);
         when "1101" =>
           B_P(2 downto 0) <= A_P(15 downto 13);
           B_P(15 downto 3) <= A_P(12 downto 0);
         when "1110" =>
           B_P(1 downto 0) <= A_P(15 downto 14);
           B_P(15 downto 2) <= A_P(13 downto 0);
         when "1111" =>
           B_P(0) <= A_P(15);
           B_P(15 downto 1) <= A_P(14 downto 0);
         when others =>
           B_P <= A_P;
       end case;
     end process; --End SHIFT
   end RTL;

```

(continued)

mized variants. Figure 1 gives you an idea of what I mean.

## ROLL YOUR OWN RISC

Xilinx's AllianceCORE program recognizes one of the most interesting and potentially profound trends—third-party IP cores that boost FPGA design options with a buy-versus-make alternative.

As illustrated in Table 1, IP cores include CPUs, peripherals, and support logic. Buy what you need and stitch it together with your own application-specific functions. Voilà, instant FPGA system on a chip.

The idea of putting a CPU in an FPGA is particularly intriguing, inspiring one to question conventional design-partitioning wisdom. Just what is the difference between hardware and software anyway, and which should do what?

One of the earliest references (1994) I found on the FPGA-CPU subject was an application note written by Ken Chapman of Xilinx [1]. Constrained by yesterday's silicon, he came up with a minimalist multi-MIPS programmable state machine in a mere 50 CLBs.

Ken reports that simply porting the original design to new chips boosts performance 3×. Additionally, optimizing to take advantage of the new synchronous dual-port RAM cuts logic and doubles performance.

That's a 6×+ performance boost in three years—I love this business! It's the march of silicon that makes IP cores, formerly an academic curiosity, increasingly viable for volume commercial application.

Today, designers can choose from a wide selection of mainstream controller cores (standbys like the '51, 6502, and Z80). These are obviously useful when compatibility with existing software and tools is paramount, as when sweeping a board's worth of logic into one chip.

There are also brand new cores emerging. Freed of compatibility baggage and reflecting the latest architecture trends, these devices offer great potential for new designs.

Check out the V8 from VAutomation diagrammed in Figure 2. It's a

midrange 8-bit, 64-KB address load/store machine with a concise set (33 opcodes, 4 addressing modes) of variable length (1–3 byte, 1–7 clock) instructions.

Delivered in either VHDL or Verilog, the V8 comes with a quiver of peripherals including timer, UART, I<sup>2</sup>C, and DMA. USB and 1394 cores are available for extra cost. Before you go overboard, keep in mind the CPU core alone consumes 268 Xilinx CLBs (remember their Spartans ranged from 100 to 784 CLBs).

Once a CPU is in an FPGA, the temptation is to tweak it with application-specific add-ons. The V8 accommodates such desires with two spare opcodes that can handle operations like 512-/1024-bit math ops for cryptography, multiply and accumulate with saturation for DSP, block move, search and sort for networking, and so on. Multiple register sets can be tailored to provide superior interrupt response.

The TX40x series from T7L takes customization to the limit. The cores shown in Table 2 comprise variants of a base architecture in which the processing power (ALU width, number of registers), program and data address space, and instruction set are all scalable.

Performance is impressive with Harvard design and a three-stage pipeline executing fixed-length single-word (i.e. 10–24 bits, depending on the model) instructions at up to 33 MHz. Entry-level models (e.g., 8-bit TX401) are in the 120–170-CLB range, well within the means of today's FPGAs.

But, cores need all the support and debug tools of a regular CPU. Both VAutomation and T7L cover the basics with PC-based software tools (ASM and C), evaluation boards (see Photos 2a and 2b, respectively), and test and verification info. Obviously, tools for IP cores have to measure up for the concept to take off.

## LET'S MAKE A DEAL

The idea of IP-based FPGA system-on-chip is starting to get real, but don't get too starry-eyed. Further advances in FPGAs are required for even better price/performance tradeoffs, especially when it comes to on-chip RAM.

Tools will mature and improve as they deal with the increasing design and verification challenges posed by

### Listing 1—continued

```
b) library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_arith.all;
   entity barrel is
   port (S:  in  STD_LOGIC_VECTOR (3 downto 0);
        A_B: in  STD_LOGIC_VECTOR (15 downto 0);
        B_P: out STD_LOGIC_VECTOR (15 downto 0);
   end barrel;
   architecture RTL of barrel is
   signal SEL1,SEL2: STD_LOGIC_VECTOR (1 downto 0);
   signal C:         STD_LOGIC_VECTOR (15 downto 0);
   begin
     FIRST_LVL: process (A_P, SEL1)
     begin
       case SEL1 is
         when "00" => -- Shift by 0
           C                                     <= A_P;
         when "01" => -- Shift by 1
           C(15)                                <= A_P(0);
           C(14 downto 0)                       <= A_P(15 downto 1);
         when "10" => -- Shift by 2
           C(15 downto 14)                     <= A_P(1 downto 0);
           C(13 downto 0)                       <= A_P(15 downto 2);
         when "11" => -- Shift by 3
           C(15 downto 13)                     <= A_P(2 downto 0);
           C(12 downto 0)                       <= A_P(15 downto 3);
         when others =>
           C                                     <= A_P;
       end case;
     end process; --End FIRST_LVL
     SECND_LVL: process (C, SEL2)
     case SEL2 is
       when "00" => -- Shift by 0
         B_P                                     <= C;
       when "01" => -- Shift by 4
         B_P (15 downto 12) <= C(3 downto 0);
         B_P (11 downto 0)  <= C(15 downto 4);
       when "10" => -- Shift by 8
         B_P (7 downto 0)   <= C(15 downto 8);
         B_P (15 downto 8)  <= C(7 downto 0);
       when "11" => -- Shift by 12
         B_P (3 downto 0)   <= C(15 downto 12);
         B_P (15 downto 4)  <= C(11 downto 0);
       when others =>
         B_P                                     <= C;
     end case;
     end process; --End SECND_LVL
     SEL1 <= S(1 downto 0);
     SEL2 <= S(3 downto 2);
   end rtl;
```

evermore gates. It won't be easy, but technology will inexorably march on.

Nontechnical IP business and legal issues mustn't be overlooked. For instance, how should a core be sold? Fixed price, royalties, by site, by project, percent of die size? Source code or netlists? Service, support, customization?

Core prices may vary from a few thousand (netlist only) to hundreds of thousands (source, customization services, etc.). For now, all IP deals involve a lot of individual negotiation.

What about patents, especially in the case of IP clones of existing chips? One article I read describes how Cadence is establishing design centers in offshore lawyer-free zones [2]. Check your IP fine print carefully.

As questions are answered, the pace of FPGA system-on-chip development will quicken. So, stay tuned....

*Tom Cantrell has been working on chip, board, and systems design and marketing in Silicon Valley for more*



TX Series	Data Width (bits)	Data Address Width (bits)	Inst. Width (bits)	Inst. Address Width (bits)	#Inst.
TX401	8, 10	8, 10	10, 12	10, 12	47+
TX402	8, 10	10, 12, 14	12, 16	12, 14, 16	80+
TX403	12, 16	12, 14, 16	12, 16	14, 16, 18	47+
TX404	12, 16	12, 16	12, 16	14, 16, 18	80+
TX405	12, 16	12, 14, 16	12, 16	14, 16, 18	140+
TX406	24, 32	24, 32	16, 24	18, 24	80+
TX407	24, 32	24, 32	16, 24	18, 24	140+

Table 2—Shades of bit-slice! The T7L architecture is fully scalable in terms of processing power and address space.

than ten years. You may reach him by E-mail at [tom.cantrell@circuitcellar.com](mailto:tom.cantrell@circuitcellar.com), by telephone at (510) 657-0264, or by fax at (510) 657-5441.

## REFERENCES

- [1] K. Chapman, *Dynamic Microcontroller in an XC4000 FPGA*, Xilinx, App note, 1994, [www.xilinx.com/appnotes/microcnt.pdf](http://www.xilinx.com/appnotes/microcnt.pdf).
- [2] R. Goering, "Can Scotland build system-chip haven?," *EE Times*, December 15, 1997.

## SOURCES

### Spartan series FPGAs, Alliance CORE

Xilinx Inc.  
2100 Logic Dr.  
San Jose, CA 95124  
(408) 559-7778  
Fax: (408) 559-7114  
[www.xilinx.com](http://www.xilinx.com)

### 16450-type UART

Comit Systems  
1250 Oakmead Pkwy., Ste. 210  
Sunnyvale, CA 94088  
(408) 988-7966  
Fax: (408) 988-2133  
[www.comit.com](http://www.comit.com)

### HC DL controller

CoreEl Microsystems  
46750 Fremont Blvd., Ste. 208  
Fremont, CA 94538  
(510) 770-2277  
Fax: (510) 770-2288  
[www.coreel.com](http://www.coreel.com)

### PCMCIA interface library

Digital Objects Corp.  
3550 Mowry Ave., Ste. 101  
Fremont, CA 94538  
(510) 795-2212  
Fax: (510) 795-2219  
[www.digitalobjects.com](http://www.digitalobjects.com)

### 16450-type UART

Memec Design Services

819 S. Dobson Rd., Ste. 203  
Mesa, AZ 85202  
(602) 491-4311  
Fax: (602) 491-4907  
[www.memecdesign.com](http://www.memecdesign.com)

### USB function, hub controllers,

#### Galileo FSxi

Mentor Graphics  
Inventra Business Unit  
1001 Ridder Park Dr.  
San Jose, CA 95131-2314  
(503) 685-8000  
Fax: (408) 451-5690

### Synplicity, Synplify

Synplify, Inc.  
624 E. Evelyn Ave.  
Sunnyvale, CA 94086  
[www.synplify.com](http://www.synplify.com)

### TX40x

T7L Technology, Inc.  
780 Charcot Ave.  
San Jose, CA 95131  
(408) 321-9728  
Fax: (408) 383-9613  
[www.t7l.com](http://www.t7l.com)

### V8

VAutomation  
20 Trafalgar Sq., Ste. 443  
Nashua, NH 03063  
(603) 882-2282  
Fax: (603) 882-1587  
[www.vautomation.com](http://www.vautomation.com)

### 8051,6502,Z80-compatible CPU,

#### 146818-type RTC

Virtual IP Group  
1094 E. Duane Ave., Ste. 211  
Sunnyvale, CA 94086  
(408) 733-3344  
Fax: (408) 733-9922  
[www.virtualipgroup.com](http://www.virtualipgroup.com)

## I R S

425 Very Useful  
426 Moderately Useful  
427 Not Useful

# PRIORITY INTERRUPT

## Design98—A Marketer's PICnic



If you hang around me long enough, you learn that I tend to use a lot of metaphors when I'm trying to make a point. Invariably, when I'm talking about the evolution of embedded control, I eventually get to a comment that, someday, even toasters will contain microprocessors. Of course, I've been saying this to emphasize how far embedded processors have come both in application and cost effectiveness. In truth, it's just a metaphor and I've never really looked to see if any toasters use them.

Well, I guess I have to coin a new metaphor. I've finally seen a design for a microprocessor-controlled toaster oven. Mind you, this is no schlock control scheme. It's a serious demonstration of surface-mount technology, with a platinum RTD temperature sensor, 2 × 8 LCD display, rotary-encoded input dial, and RS-232 interface!

No, I didn't whip this up in the cellar. It was one of the designs submitted among the tremendous volume of entries we received for Design98, the Circuit Cellar/Microchip PIC design contest. Probably the best part about these contests is the richness of design ideas that are presented. Here was a group of people who had solved engineering problems and they wanted to tell the world about it.

And tell about it they did. We received projects on RF remote control, video digitizing, motor controls, image scanning, security keys, energy management, electronic games, a single-chip Internet server, smart switches, I/O expanders, data loggers, rocket telemetry, fuzzy-logic buck conversion, a taxi meter, and assorted transmitters, receivers, programmers, interpreters, etc., etc., etc. And, oh yes, there was a toaster-oven controller.

While everyone could not be a winner, they all were worthy of winning. And no doubt, they'll be rewarded in the marketplace. Of course, certain projects really caught my eye. In the *necessity is the mother of invention* category, there was the Great Highland Bagpipe Chanter. Apparently, learning to play the bagpipe is a demanding physical task. It's equally demanding on anyone who's around you when you're practicing. The only volume control on a bagpipe is distance! With the addition of a few buttons and a 12" PVC pipe, this ingenious entrant made a PIC-based electronic bagpipe simulator so his students could practice without creating a riot.

In the proverbial *10 pounds in a 5-pound bag* category, we had a couple graphing data loggers. The first was a weather monitor with a built-in 128 × 240 LCD. It monitored, stored, and displayed pressure and temperature as a continuously updated scrolling graph showing the present conditions as well as the previous 48 hours of weather data. The second project was an *x-y* graphing data logger with eight 12-bit ADC inputs, 20 KB of data space, and a 200-hour battery life. Using a Casio graphing calculator as the display (64 × 128 LCD), the combined system performed sophisticated analytical and statistical math processing on the analog data.

In the *it's better to see you* category, I loved the electronic automobile sun-visor project. This one answers the question, what do you do (besides swear a lot) when the sun isn't blocked by the usual flip-down sun visors? This circuit uses two CCD linear arrays to continuously track the sun's *x-y* position. With that information and some neat calculations, it automatically blocks the blinding sun from reaching the driver's eyes by darkening the appropriate pixels on a 24 × 64 pixel (0.4" × 0.6" each) optically clear automotive LCD sun visor. When can I get one?

Finally, in the *why didn't I think of that* category, there was an X-10 Remote Temperature Sensor. The circuit used a Dallas digital thermometer and 2 × 16 LCD (as a local display) combined with a TW523 X-10 transceiver as you might typically expect. Without using X-10 extended data, however, designers tend to resort to using a whole bunch of house and unit codes to represent the wide range of potential temperatures. This circuit relied instead on the host controller and a single house code. The sensor decodes two consecutive On commands to ask a specific comparison temperature value. The sensor responds with an On command if the comparison temperature is greater than or equal to actual temperature or an Off if it's less. It takes a few commands to zero in on the measured temperature, but this technique allows many sensors.

Some of the Design98 entries were elaborate; others ingeniously simple. For Microchip, it only confirmed their assertion that cost-effective 8-bit processing ultimately expands all the potential applications. As for us, we've got a boatload of great designs and a whole new group of potential authors. Our task will be turning many of these entries into published projects on our Web site and in the magazine. Congratulations to all the entrants for a job well done.



steve.ciarcia@circuitcellar.com