CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

SIGNAL PROCESSING

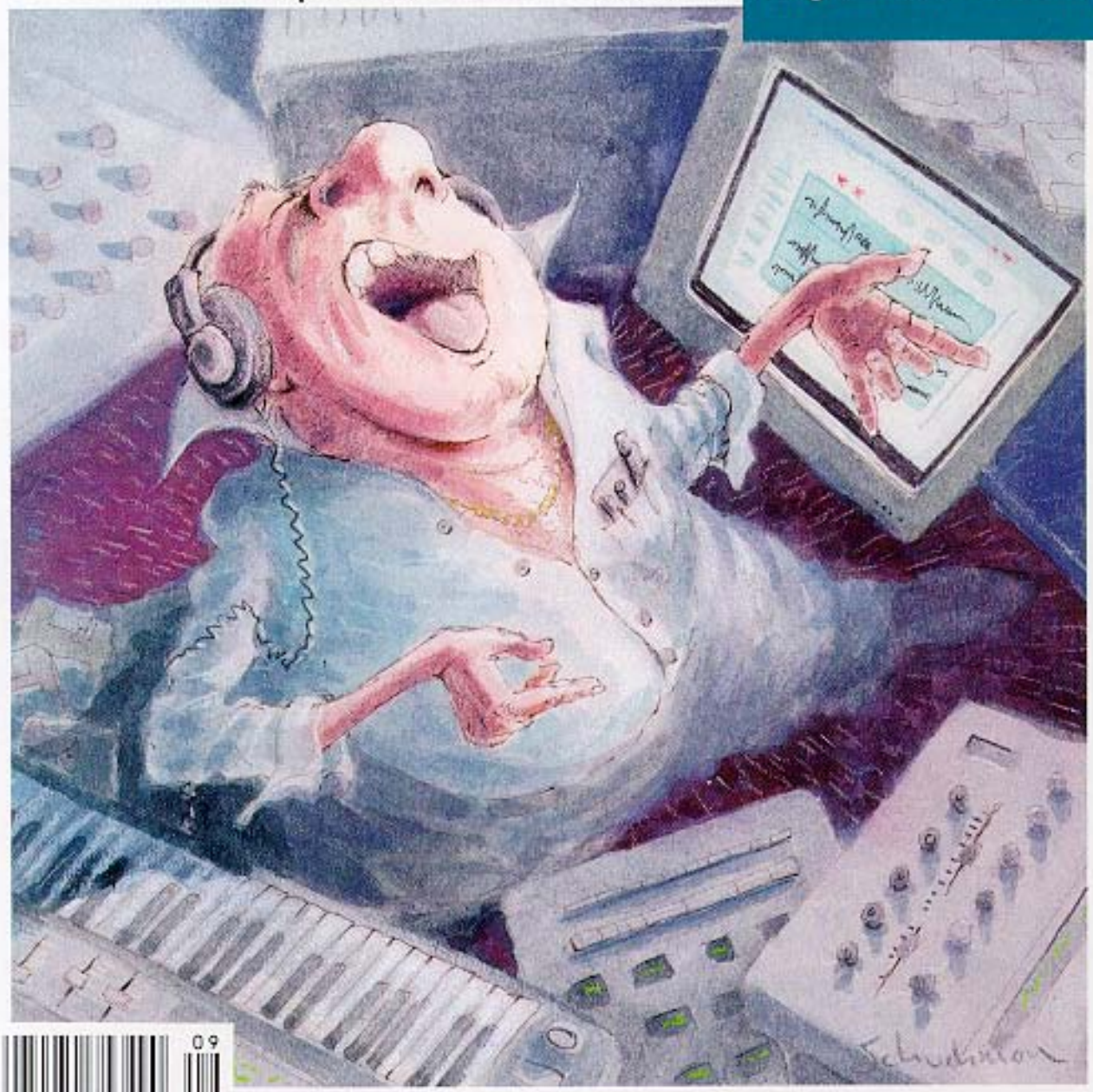
Will a RISC Processor Work for DSP?

Analyze Functions with your PC

Neural Network Basics

Two New and Tiny One-time Programmable Processors

September 1993 — Issue #38



09
0 74470 75349 0

$3.95 U.S.
$4.95 Canada

EDITOR'S INK

Mainstream Processing

The proliferation of digital signal processing in today's mainstream society is truly astounding. True, DSP has been in the labs and in high-end equipment for many years, but the true test of a technology's viability is when it is integrated into mass-produced consumer equipment. Witness the enormous success of the compact disc over the last 10 years. It has revolutionized quality music reproduction and has breathed new life into the recording industry. Signal processors that add concert-hall-like ambiance to recorded sound have also taken off, thanks completely to DSP.

I've always considered where I live to be somewhat "in the sticks." We live on a dirt road in a sparsely populated part of town. Imagine my surprise when I asked the telephone lineman down the street what he was doing and he responded that he was activating a fiber-optic substation not 2500 feet from my house. When (and if) SNET decides to implement ISDN or some other digital service, we'll be close enough to a substation that subscribing to the service should be painless. In the meantime, we have the benefit of crystal-clear telephone service, thanks again to DSP.

We start off our signal processing issue with the question, can a RISC processor do DSP as well as a dedicated DSP chip? In one of his "having pun with a processor" articles, professor Michael Smith implements an FIR filter on an Am29050 RISC processor, describes the development cycle he went through, and explores its potential in DSP applications.

Next, we add some real-world interfaces to an off-the-shelf IBM PC I/O card and both control function generator chips and analyze incoming audio signals, all using a desktop IBM PC.

In our third feature article, we turn our attention to neural networks and how they can be used in everyday life. A single program can be written and used for many situations by simply retraining that program for each instance.

In our columns, Ed's embedded '386SX system gains a BIOS extension to handle hardware the original PC designers never dreamed of. He also passes on some valuable advice he learned in the trenches while debugging an old project: Jeff has fitted last month's optical ID card reader with some glasses and boosts its resolution by an order of magnitude. Tom checks out a pair of new one-time programmable processors that are small in size and price. John describes several techniques for adding lots of memory to small microcontrollers. Finally, Russ has searched his database for spread-spectrum-related patents and has found some interesting tidbits.

CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

FOUNDER/EDITORIAL DIRECTOR
Steve Ciarcia

EDITOR-IN-CHIEF
Ken Davidson

TECHNICAL EDITOR
Michael Swartzendruber

ASSOCIATE EDITOR
Robert Rojas

ENGINEERING STAFF
Jeff Bachiochi & Ed Nisley

WEST COAST EDITOR
Tom Cantrell

CONTRIBUTING EDITORS
John Dybowski & Russ Reiss

NEW PRODUCTS EDITOR
Harv Weiner

ART DIRECTOR
Lisa Ferry

GRAPHIC ARTIST
Joseph Quinlan

CONTRIBUTORS:
Jon Elson
Tim McDonough
Frank Kuechmann
Pellervo Kaskinen

Cover illustration by Bob Schuchman
PRINTED IN THE UNITED STATES

PUBLISHER
Daniel Rodrigues

PUBLISHER'S ASSISTANT
Susan McGill

CIRCULATION COORDINATOR
Rose Mansella

CIRCULATION ASSISTANT
Barbara Maleski

CIRCULATION CONSULTANT
Gregory Spitzfaden

BUSINESS MANAGER
Jeannette Walters

ADVERTISING COORDINATOR
Dan Gorsky

CIRCUIT CELLAR INK. THE COMPUTER APPLICATIONS JOURNAL (ISSN 0896-8985) is published monthly by Circuit Cellar Incorporated, 4 Park Street. Suite 20, Vernon. CT 06066 (203) 875-2751. Second class postage paid at Vernon. CT and additional offices One-year (12 issues) subscription rate USA and possessions $21 95, Canada/Mexico $31.95, all other countries $49.95. All subscription orders payable in U.S funds only, via international postal money order or check draw" on U.S bank. Direct subscription orders and subscription related questions to The Computer Applications Journal Subscriptions, P 0 Box 7694, Riverton, NJ 08077 or call (609) 786-0409.
POSTMASTER: Please send address changes to The Computer Applications Journal, Circulation Dept.. P.O. Box 7694, Riverton, NJ 08077

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

INSIDE ISSUE 38

READER'S INK

The Sample Game Rules

As a hacker/entrepreneur, I read Steve Ciarcia's comments about the "sample game" with much interest ("Steve's Own INK," February 1993). As a field application engineer for Bell Industries (an electronics distributor) I can sympathize with his plight. I disagree with his solution, on a couple of counts.

First, understand that a distributor has direct and overhead expenses, just like your business. Every time the phone rings, it costs a distributor five bucks just to say hello. If the distributor is to break even *(not* make a profit) he must sell at least fifty dollars worth of parts.

What Steve is suggesting is that you pull the wool over the distributor's eyes (quite easy to do, unfortunately), do a bluff, and get some "free" samples to boot. Of course, it costs the distributor five bucks to answer the phone, three bucks to put the parts in the box, ten bucks to ship the product, plus the cost the distributor paid for the product, but hey! What's $18 among friends?

Fortunately, some distributors are getting more in step with the real world, just like Maxim and Dallas Semiconductor. Instead of the $350 line item minimums, credit applications, and legal agreements that only lawyers can read, try calling the account development group of your local distributor.

As an example, Bell Industries has a phone number (800-BUY-BELL) which allows you to use your credit card and requires only a $50 purchase. Do not expect technical support from these folks; their only function is to answer questions about price and take your order.

Most local distributors will also allow you to come in and pick up data books or data sheets, as long as you don't abuse it. If you are truly working toward a product, and not just fooling around, ask to talk to the FAE. As in any job, there are good ones and bad ones, some who have never actually built a circuit with their bare hands, and some who are up until 3 A.M. trying to get the damn thing to work, just like you and me.

Just don't expect miracles from distribution. Every company I know of is trying to cut costs, and that means those employees "lucky" enough to still be around end up timesharing between two or three jobs. I think you'll find there are distributors with decent salespeople, but the decent salespeople are not likely to go running after a five- or ten-piece order. You have had some unfortunate experiences with distribution, and I'm sure your opinion is justified. Just don't throw out the barrel because of one or two rotten apples.

David Riness
Cypress, Calif.

Please understand that Steve did not intend to let the cat out of the bag on how to get unlimited, free components from the distribution channel. He was merely illustrating what happens when the focus shifts from "long-term engineering development and support" to "nothing but the quota matters, don't *bother* me *unless you have a $10,000 order" mentality.*

It seems the whole industry suffered when it was taken over by the suits and bean counters. These folks forgot (or never knew) what it's like to develop a new product on a limited budget, carve out a market for this product, and after the course of a few quarters (or years), the payback for all of the efforts begins.

Have the policy makers at BigWorld Electronics forgotten that HP and Apple were started by just two engineers with stars in their eyes! Selling into these accounts is probably enough to have sales reps wrestling at the door. But what if these guys were trying to start these ventures in today's climate! Would they get parts! Would there be anyone that could answer an even minor technical question! Would the "I am only interested- in sure things" mentality keep these guys from getting the attention they need!

You see, you *don't know where the next Apple is going to come from. And that is the risk you must assume if you are going to be involved in this business. Hey, what's $18 among friends! Okay, that's justifiable. But, hey, make the guys at NewStartUp Inc. upset with the whole way you do things, and maybe they'll never call you after they are placing those kilobuck orders. And don't forget that you guys provide service to engineers, not bankers.*

Granted, we should talk to the FAE. But truthfully, aren't your attentions given due to the same "sales potential" criterion! We have had some close relationships with sales reps and FAEs, and in our experience the FAE is a limited resource under the direct control of the sales department. If Bob the salesman says this must be done, then out of necessity, the question asked by NewStartUp Inc. goes to *the back* burner.

We are pleased to hear that new methods for component purchase are being explored to keep the whole industry from becoming an "old boys club." This is a step in the right direction, but now your organization has to make it easier to learn about it, not just buy.

As you can see this issue is an important one. The very forces of innovation are at stake. Let's hope we make the right moves, or we may all pay dearly for the wrong turns.
-Editors

READER'S INK

More Chilling Thoughts

With regard to the question of how wind chill factors were determined ("Reader's INK," The *Computer* Applications Journal, June 1993), I read somewhere several years ago that these factors were determined empirically. The experiment was to take sausage-shaped bags filled with water (about the size of an adult forearm) and expose them to various subfreezing air temperatures and wind speeds.

At each temperature to be tested, the reference test was at zero wind speed, with the time to freezing measured. Then the time to freezing was measured at various combinations of temperature and wind speed. If, for instance, the time to freezing at 20°F and 15 MPH was the same as 0°F and 0 MPH, the wind chill at 20°F and 15 MPH was said to be 0°F.

I'm sure much interpolation and extrapolation was done to fill out the tables. The formula Dana Romero lists looks like many such equations I saw when I took Heat Transfer in college-the result of some deranged curve-fitting program left to run on a VAX (then!) overnight.

The method used to determine the wind chill numbers does bring up questions as to their validity, or at least applicability. As one weather reporter said, these numbers are only valid if you're naked in the shade. I think the numbers are popular because they let people brag about how tough they are-to be out when the wind chill was 45 below zero.. .

Curt Wilson
Northridge, Calif.

We Want to Hear from You

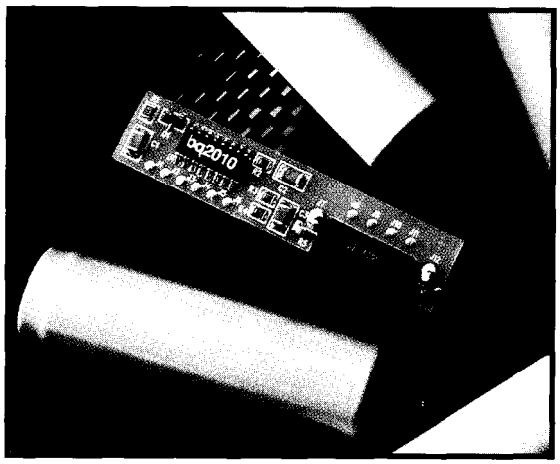
We encourage our readers to write letters of praise, condemnation, or suggestion to the editors of the Computer Applications Journal. Send them to:

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

NEW PRODUCT NEWS

Edited by Harv Weiner

BATTERY CAPACITY MONITOR



The first integrated solution for comprehensive monitoring of battery capacity has been announced by Benchmarq Microelectronics. The **bq2010** Gas Gauge IC provides accurate and repeatable measurement of the available charge in NiMH and NiCd rechargeable batteries.

The bq2010 directly displays available battery capacity using LEDs and does not require an external processor, allowing designers to upgrade their systems to smart battery packs without having to redesign the product. The battery's charge state and other battery-management information can also be provided to a controller through a single-pin serial port for a customized display.

Because the bq2010 is integrated within the battery pack, critical capacity information is not lost with each battery removal. External charge stations need not provide tightly regulated charge currents and/or intelligence to update the available battery charge and capacity.

The bq2010 supports a simple, single-line, bidirectional, serial link to an external processor, and can output battery information in response to external commands over this serial link. Internal registers include available charge, temperature, capacity, battery ID, battery status, and the settings of the device's programming pins.

The bq2010 is packaged in a small, 16-pin,150-mil SOIC. It, and all of its necessary external components, can fit in the crevice between two AA cells. This eliminates the need to retool the plastics for the battery pack to accommodate the capacity monitor.

The bq2010 Gas Gauge IC sells for $5.70 for a 16-pin narrow DIP and $5.90 for the 16-pin SOIC package (1k quantities). A development kit is available for $75.

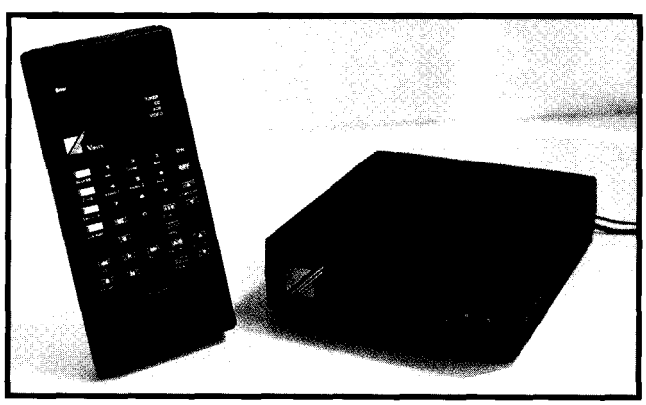
Benchmarq Microelectronics, Inc.
2611 Westgrove Dr., Ste. 109 • Carrollton, TX 75006
(214) 407-0011 . Fax: (214) 407-9845 **#500**

MULTIROOM ACCESS REMOTE CONTROL

The MARC (Multiroom Access Remote Control) System enables the control of virtually any device from any room in a house without wiring. Four kinds of infrared controlled audio/video equipment can be operated with one remote control at distances up to 125 feet.

The MARC System can also operate up to 256 X-10 Powerhouse light, appliance, and security modules. For additional security, each system has its own identification code that is transferred with every button pressed. The code can be established during initial setup and changed as often as necessary. Expansion options include an advanced security system, energy management, telephone dial-in/dial-out support, and personal computer interface modules to allow the design of a customized home automation system.

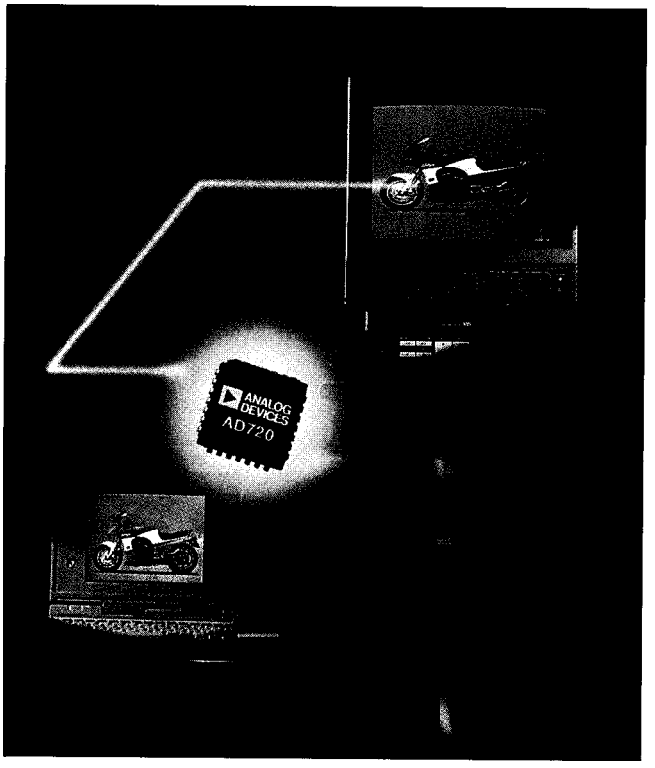
The MARC System is composed of a universal remote control and a receiver. Depending on the button selected, the receiver will do one of the following: convert the RF signals into infrared signals to control audio/video equipment, send commands over the power lines for managing X-10 modules, or operate any of the optional expansion modules.



The MARC receiver is designed to be placed behind or beside the audio/video components. Because the system is wireless, the equipment can also be installed in an out-of-sight cabinet. The emitter is placed near or over the infrared sensor window on the front panel of the device to be controlled.

Vaux Electronics, Inc.
5310 Logan Ave. South
Minneapolis, MN 55419
(612) 920-5037 • Fax: (612) 925-5175 **#501**

NEW PRODUCT NEWS



RGB-TO-NTSCIPAL ENCODER

The industry's first analog RGB-to-NTSC/PAL encoder provides video system designers with a high-performance, fully calibrated, single-IC solution that does not require discrete low-pass filters or delay lines. The AD720 from Analog Devices features a composite video output, a differential gain of 0.1%, and a differential phase shift of 0.1". This level of performance results in NTSC/PAL video outputs that are capable of generating smear-free reverse type (with text as small as 9 points) in applications such as PC video cards, multimedia systems, CATV converter boxes, and other video or imaging systems.

The AD720 converts the red, green, and blue video signals into their corresponding luminance (baseband amplitude), chrominance (subcarrier amplitude and phase), and composite (combined luminance and chrominance) video signals. The AD720 produces superior picture quality, largely due to the use of thin-film resistors in the RGB-to-YUV matrix, calibrated on-board low-pass filters and delay line, and digitally generated quadrature signals. In addition, the AD720 uses multiple ground and power supply pins to lower internal package impedance, which can help prevent crosstalk and signal feedthrough. This obviates the need for external adjustments.

The integration provided by three on-chip, low-pass filters and a delay line results in enhanced ease of use, especially when compared to other discrete video encoding methods. No analog design experience is necessary to implement the AD720 into a high-performance system. Two 4-pole filters bandlimit the U/V color-difference signals to 1.2 MHz prior to subcarrier (color) quadrature modulation. A third 3-pole filter follows the modulator and limits the harmonic content of the user-selected NTSC or PAL output. An on-board 170-ns delay function provides precompensation for delays in the filters used to decode the NTSC or PAL signal in television receivers.

The separate luminance, chrominance, and composite output voltages are DC coupled, thereby providing S-Video output. The AD720 is capable of driving 75-ohm reverse-terminated loads through the use of on-board gain-of-two output amplifiers, which amplify the output voltage's signal amplitudes to twice that of NTSC and PAL standards. The AD720 typically dissipates 200 mW (when powered with ±5-V supplies), and features a power down mode that reduces consumption to less than 50 mW. The power down mode is logic selectable and can be done whenever the chip's encoding function is not in use. All logic inputs are standard CMOS level compatible. The AD720 is housed in a 28-pin plastic leaded chip carrier (PLCC) and sells for $18.39 in quantity.

Analog Devices, Inc . 181 Ballardvale St. . Wilmington, MA 01887 . (617) 937-1428 . Fax: (617) 821-4273 **#502**

LOW-COST PROGRAMMER

Single Chip Solutions has announced a low-cost programmer that connects to an IBM PC or compatible through a serial port (COM1 or COM2). Power for the PGMHC05 is provided by a 9-VDC wall transformer. All programming voltages are generated and regulated on the programmer board. The programmer is supplied with a Windows interface that supports programming and verifying parts, as well as uploading data from programmed parts. It is also compatible with the B U RN 0 5 software, available from Motorola's Freeware BBS.

The programmer is available as a kit or assembled. Prices start at $100 for the kit. Programmers for the MC68705 and MC68701 are also available.

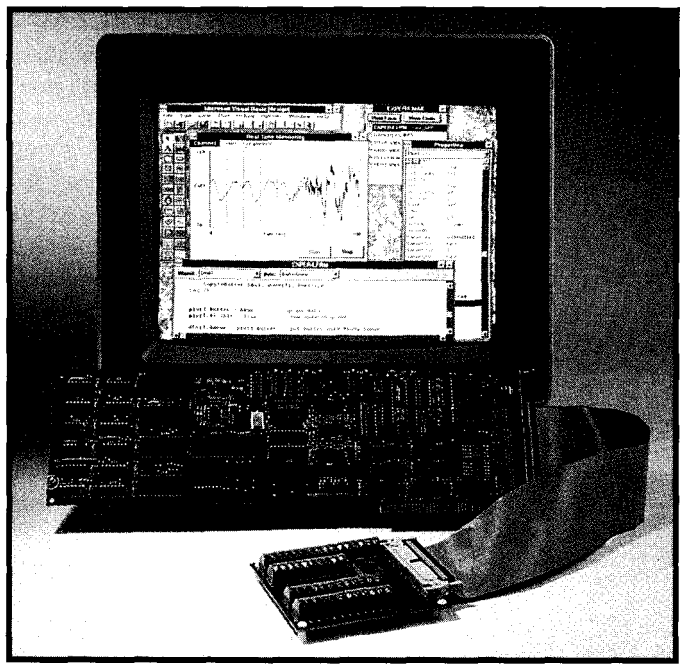
Single Chip Solutions
P.O. Box 680 . New Hartford, CT 06057-0680
(203) 496-7794

#503

NEW PRODUCT NEWS

LOW-COST DATA ACQUISITION

DataAcq-EZ, a low-cost solution for Windows-based data acquisition, has been announced by Data Translation. Three data acquisition boards with varying resolutions and speed are integrated with the **VB-EZ** data acquisition programming tools to easily create powerful graphical and scientific applications using Visual Basic.



The **DT01-EZ** board features 12-bit resolution, 27.5-kHz throughput, analog and digital I/O; the **DT21-EZ** features 50-kHz speed; and the **DT16-EZ** increases resolution to **16** bits. Each board also contains two independent 12-bit D/A converters, two 8-line digital I/O ports, software and/or external triggering with on-board clocking, and can transfer data using DMA. The DT2 1 -EZ also offers random channel scan, and a 16-element channel/gain list.

VB-EZ optimizes Visual Basic for real-world data acquisition by providing a Data Acquisition Custom Control. Using this Control and the Visual Basic constructs of Properties, Events, and Methods, simplifies hardware configuration by insulating the user from the complexity of the I/O boards. A High-speed Custom Control for plotting quickly generates engineering and scientific graphs that display acquired data in real time.

VB-EZ supports all analog and digital I/O functions of the Data-Acq-EZ boards, and includes subroutines and function procedures that perform buffer and data management pertaining to the specific board installed. Context sensitive help is available at all times, supporting all elements of program development.

The DT01-EZ board sells for $595; the DT21-EZ for $795; and the DT16-EZ for $995. The VB-EZ Visual Basic Programming Tools sells for $195, and the STP-EZ Screw Terminal Panel sells for $99.

Data Translation
100 Locke Dr.
Marlboro, MA 01752-I 192
(508) 481-3700
Fax: (508) 481-8620

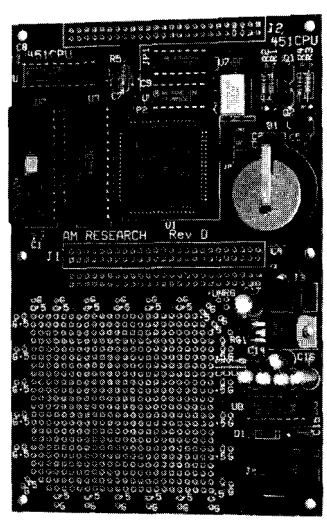
#504

LOW-COST MICROCONTROLLER

Two low-cost controllers have been introduced by AM Research. The **amr451LC** and **amr552LC** are embedded controllers which incorporate CMOS supersets of the popular 805 1. These versatile microcontrollers contain an RS-232 serial port and +5-V regulation. Other standard features include 8/32K RAM and ROM sockets; 56 and 48 I/O lines respectively; dual pulse width modulators; an g-channel, 8- or 10-bit A/D converter; a mailbox port; and a watchdog timer. The 4″×6″ boards also contain five timer/counters with expanded compare functions, an I^2C port, a prototyping area, simplified expansion of up to an additional 256 bits of I/O, and a lithium battery for RAM backup.

The systems includes an extensive array of software development tools which can produce ROMmable Forth code. Other features are a full-screen editor, in-line assembler, source-level single-step debugger, decompiler, disassembler, power source, cabling, and a 300-page manual.

The amr451LC and amr552LC each sell for $99.00. The systems sell for $199.



AM Research
4600 Hidden Oaks La. • Loomis, CA 95650 • (800) 949-8051 • Fax: (916) 652-6642

#505

NEW PRODUCT NEWS

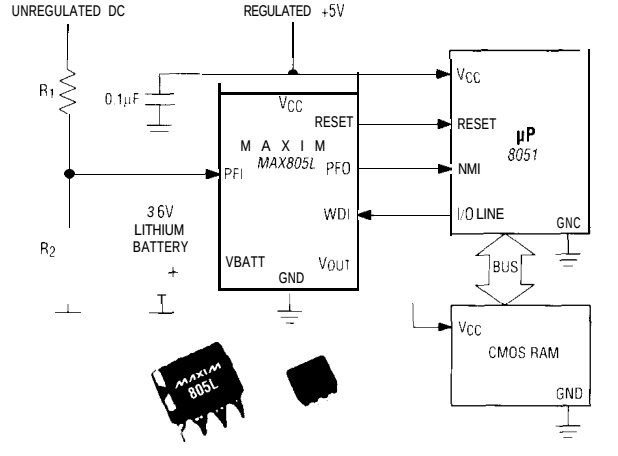
MICROPROCESSOR SUPERVISORY IC

Maxim Integrated Products has introduced the **MAX805L** and **MAX813L** microprocessor supervisory ICs. They generate active-high reset outputs (RESET) when V,. drops below 4.65 volts for all conditions of power-up, power-down, brownout, and momentary power interruptions. Active-high resets are required by many Intel microprocessors, such as the 8051 series.

The MAX8 13L has a debounced manual-reset input (MR) that is capable of generating resets on command, The MAX805L's battery switchover accommodates backup battery power for SRAM and real-time clocks. Reset pulses are 140 ms minimum, guaranteed for V_{CC} as low as 1 volt. Each device has an independent comparator/reference circuit designed to monitor a battery, a regulator input, or any other voltage. Each includes a watchdog timer that monitors software execution by issuing a reset whenever 1.6 seconds elapses without evidence of activity on any selected I/O line.

Typical uses for these devices include battery-powered computers and controllers, intelligent instruments, automotive systems, and critical microprocessor power monitoring. Both chips come in either 8-pin DIP or SOIC packages and are available in various temperature ranges. The MAX805L sells for $3.26 and the MAX813L sells for $1.61 (1k quantities).

Maxim Integrated Products
120 San Gabriel Dr. . Sunnyvale, CA 94086
(408) **737-7600** **#506**



NEW PRODUCT NEWS

OPTICAL MEASUREMENT SYSTEM

An optical system for making distance measurements has been announced by Acuity Research. This system can precisely measure any distance from 0 to 20 meters. The AccuRange 3000 operates by emitting a collimated laser beam that is reflected from the target surface. The distance to the target is converted to a frequency that may be precisely measured. The system is suitable for a wide variety of distance measurement applications that demand high accuracy and fast response time.

The AccuRange 3000 has a standard deviation of the indicated range as low as 0.25 mm. When calibrated, its absolute accuracy and long-term drift have a standard deviation of 2 mm when used with an infrared laser. The nominal 20-meter range may be extended with retroreflective targets.

The sensor is 5.5 inches long and 3 inches in diameter. It is housed in a rugged aluminum housing, and weighs in at 18 ounces. Power requirements are 5 V at 250 mA and 9-15 V at 50 mA.

Two versions are available: one with a visible red beam and the other with a near-infrared beam. The IR version has better sensitivity and lower measurement noise, but the advantage of being able to see the beam may be a more important factor in some applications.

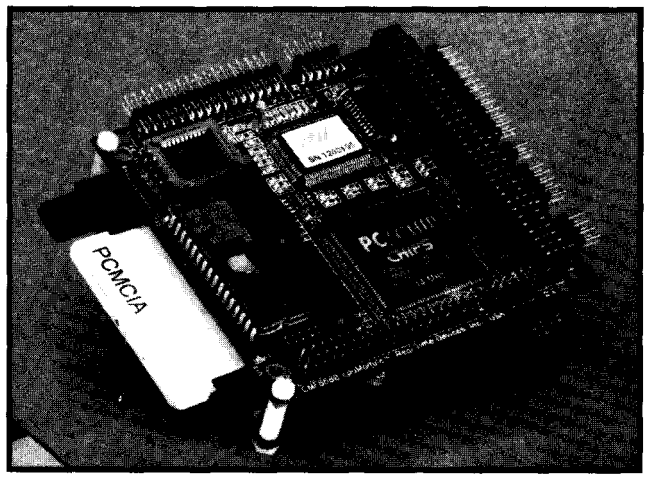
The signal that represents the target's range is available from a BNC connector on the back panel. The frequency varies from about 50 MHz (representing a zero range) to 4 MHz (representing 20 meters). A Y-pin connector is also provided for power and control signals.

A SCSI interface for the AccuRange is also available. This board will connect to any computer with a SCSI interface. It may be used as a stand-alone system, or it can be installed in an IBM-compatible PC. The board includes hardware for measuring the frequency and other outputs of the AccuRange. The sampling rate is programmable, and can go as high as 3 12,500 samples per second. The board also has precision power supplies for the sensor, an input data buffer, and an optional 2-channel, DC-motor control, and encoder sampling circuitry for use in 3-D scanning systems.

Acuity Research, Inc.
20863 Stevens Creek Blvd., Ste. 200 • Cupertino, CA 95014
(408) 252-9369 • Fax: (408) 725-1580 **#507**

NEW PRODUCT NEWS

ULTRA-COMPACT COMPUTER



Real Time Devices has announced an ultra-compact PC compatible, single-board computer measuring 3.8" x 3.6" and weighing only three ounces. The **CMF8680 cpuModule** is based on Chips & Technology's 16-bit,14-MHz, F8680 PC/Chip processor. It was designed for embedded applications requiring 100% PC software compatibility, but where traditional PCs were too large, power hungry, and unreliable. The CMF8680 is compatible with the PC/104 form factor for embedded PCs and can be expanded with any peripheral that is compliant with this standard.

The CMF8680 includes a CGA graphics display port, an LCD interface, a 16-bit IDE hard drive controller, a high-density floppy controller, and a PCMCIA interface. It features a full complement of standard PC peripherals. For software development, a keyboard, CRT, mass storage and +5-volt supply are required to make the CMF8680 a fully functional industry-standard PC.

Each CMF8680 includes a standard BIOS and DataLight ROM-DOS 5.0 kernel, and supports 1 MB of bootable, solid-state disk. Included with the system are utility programs allowing users to put their application software in ROM.

TheCMF8680 cpuModule, including 2M RAM, utility ROM, and support software sells for $895.

Real Time Devices, Inc • P.O. Box 906 • State College, PA 16804 • (814) 234-8087 • Fax: (814) 234-5218 **#508**

FEATURES

Is the Am29050 a FIR-bearing Animal?

FEATURE ARTICLE

Michael Smith

Can the Am29050 RISC processor be used where a dedicated DSP normally rules? Using a finite impulse response (FIR) filter as an example, one university professor explores the possibilities.

Digital filters are becoming increasingly popular, because they can be used to remove undesired noise from sampled data. Off-line processing is performed on stored data, and because of the amount of data, requires rapid processing. On-line filtering requires real-time processing of an incoming signal. Slow algorithms have no place in these situations.

A couple of psuedocode examples of typical digital filter algorithms are shown in Figure 1. Figure la shows an FIR (Finite Impulse Response) filter, and Figure 1b shows an IIR (Infinite Impulse Response) filter. Note that the IIR filter uses a recursive algorithm, which places some specialized restrictions on the processors used in real-time applications.

In this article, I'll concentrate on on-line FIR processing. After all, off-line data is simply yesterday's on-line data. The only difference in these two systems is that off-line processes do not have to meet stringent processing-times. What you can do in real time, you certainly can do when you can take your time. Figure 2 shows a typical on-line digital filter system.

Although not specifically designed for DSP, the Advanced Micro Devices Am29050 floating-point RISC processor has features that make it suitable for DSP. It includes a single-cycle FPU, modulo arithmetic addresssing logic, and data paths and ALUs capable of parallel operation. Its RISC architecture and internal pipelines allow you to write highly efficient programs.

I'll show you features of the Am29050 that can overcome the

problems associated with implementing FIR filters, and limitations of the chip that require special attention. I'll discuss stategies that allow the Am29050 to be used for high-speed filters. These include designing hardware for maximum throughput and the use of decimation techniques. It's possible to produce a 193-tap, linear-phase, single-processor, FIR filter with a maximum sampling rate of 2.6 µs when four-fold FIR decimation techniques are applied.

PRACTICAL APPLICATION OF AN FIR FILTER

Figure 3 shows the various stages of digital filtering. Figure 3a shows a signal that contains high- and low-frequency components. Figure 3b shows the digitized signal. Note that initially, the digitized signal faithfully records the signal. In the last part of the signal, aliasing occurred during sampling and the high-frequency signal has become a low-frequency digital signal. Once aliasing has occurred, there is very little chance it can be reversed.

The digital signal can only accurately represent the input signal if the input signal does not have signal components above the Nyquist criterion. Signals above this frequency will be aliased into lower frequencies causing distortion. Many signals are naturally limited, but wide-band noise may be present. Aliasing this noise does not necessarily introduce distortion, but it can reduce the signal-to-noise ratio. Adding a low-pass filter would prevent this problem.

This signal is then filtered with a four-tap, symmetric, low-pass FIR filter whose coefficients are 1/6, 1/3, 1/3, and 1/6. Symmetric filters reduce filter distortion. The output of the filter (Figure 3c) is produced by implementing the following equation on all input values:

$$y[n\Delta T] = \sum_{i=0}^{N-1} x[(n-i)\ AT] \times coeff\ [i]$$

N = 4; n = 1, 2, 3, . . .

```
a)  float  x[ALLDATA], y[ALLDATA]
    float  coeff[FILTERLENGTH]
    for  (n=0;  n<ALLDATA  ; n++)

        y[n]=0;
        for(i=0;  i<FILTERLENGTH ;  i++)
            y[n]+=x[n-i]*coeff[i];

b)  float  x[ALLDATA], y[ALLDATA]
    float  xcoeff[XFILTERORDER]
    float  ycoeff[YFILTERORDER]

        for(n=0 ; n<ALLDATA ;  n++)

            y[n]=0;
            for(i=0 ; i<XFILTERORDER; i++)
                y[n]+=x[n-i]*xcoeff[i];
            for(j=1 ; j<YFILTERORDER ; j++)
                y[n]+=y[n-j] * ycoeff[j];
```

Figure 1—*The Finite Impulse Response (FIR) filter (a) and the Infinite Impulse Response (IIR) filter (b) can be described by very short algorithms. The trick comes in implementing them efficiently.*

The calculations must complete between samples for a real-time filter. Long filter lengths place heavy requirements on the processor. The infinite amount of input in an on-line filter causes a storage problem for these systems and often, circular buffers must be implemented, which create considerable overhead. Finally, the analog output signal [Figure 3d] is produced by routing the output through a low-pass filter.

Decimation techniques are used to speed up digital filters. Suppose the input signal had frequency components up to 400 kHz. This would require sampling at 800 kHz (or greater) to remove aliasing. About 400 summations and multiplications must be performed between each sample. If this were attempted for a 200-tap filter, you would need a 400-MFLOPS processor! This class of processor is not cheap or easy to come by.

The output signal might only need a 50-kHz bandwidth. However the output signal is still produced with a sampling rate of 800 kHz, which is unnecessary since a 100-kHz output rate would accurately represent a 50-kHz signal. You could process one out of every eight samples and still get an accurate representation of the output signal. Using decimation, the filter's equation now looks like the following:

$$y[n\Delta T] = \sum_{i=0}^{N-1} x[(n-i)\ AT] \times coeff\ [i]$$

n = 1, 9, 17, . . .

A 200-tap filter with an 800-kHz sampling rate requires a 50-MFLOPS processor when four-fold decimation is used. The Am29050 can provide 80 MFLOPS at 40 MHz provided the pipeline can be kept full. Pipelining is the concept of splitting an operation into many steps that can be completed in parallel. With this technique, a new operation can start, or complete, every cycle. This gives pipelined processors their advantage. On a RISC processor, the pipeline is king, and any instruction that detracts from the pipeline performance is not implemented. Peak performance is obtained when the algorithm keeps the pipeline(s) full.

Although only one out of eight output values are evaluated for a decimation filter, all of the input values must be processed. For time-critical situations, a FIFO stack attached to the ADC relieves the processor of the interrupt service overhead.

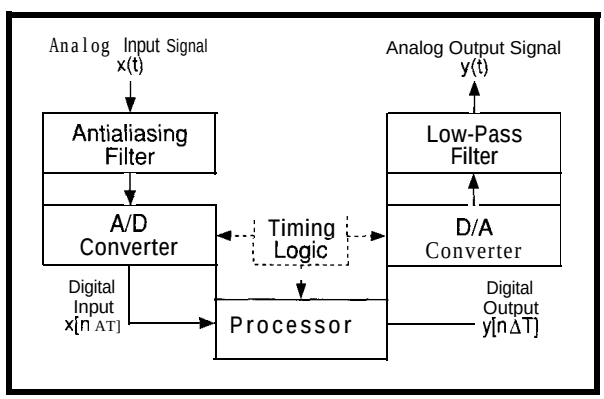


Figure 2—*A typical on-line digital filtering system.*

IMPLEMENTING AN FIR FILTER

I coded the N-(N odd) tap, symmetric FIR filter algorithm shown in the equation below in a straightforward way. I'll discuss the problems that this approach caused and how the

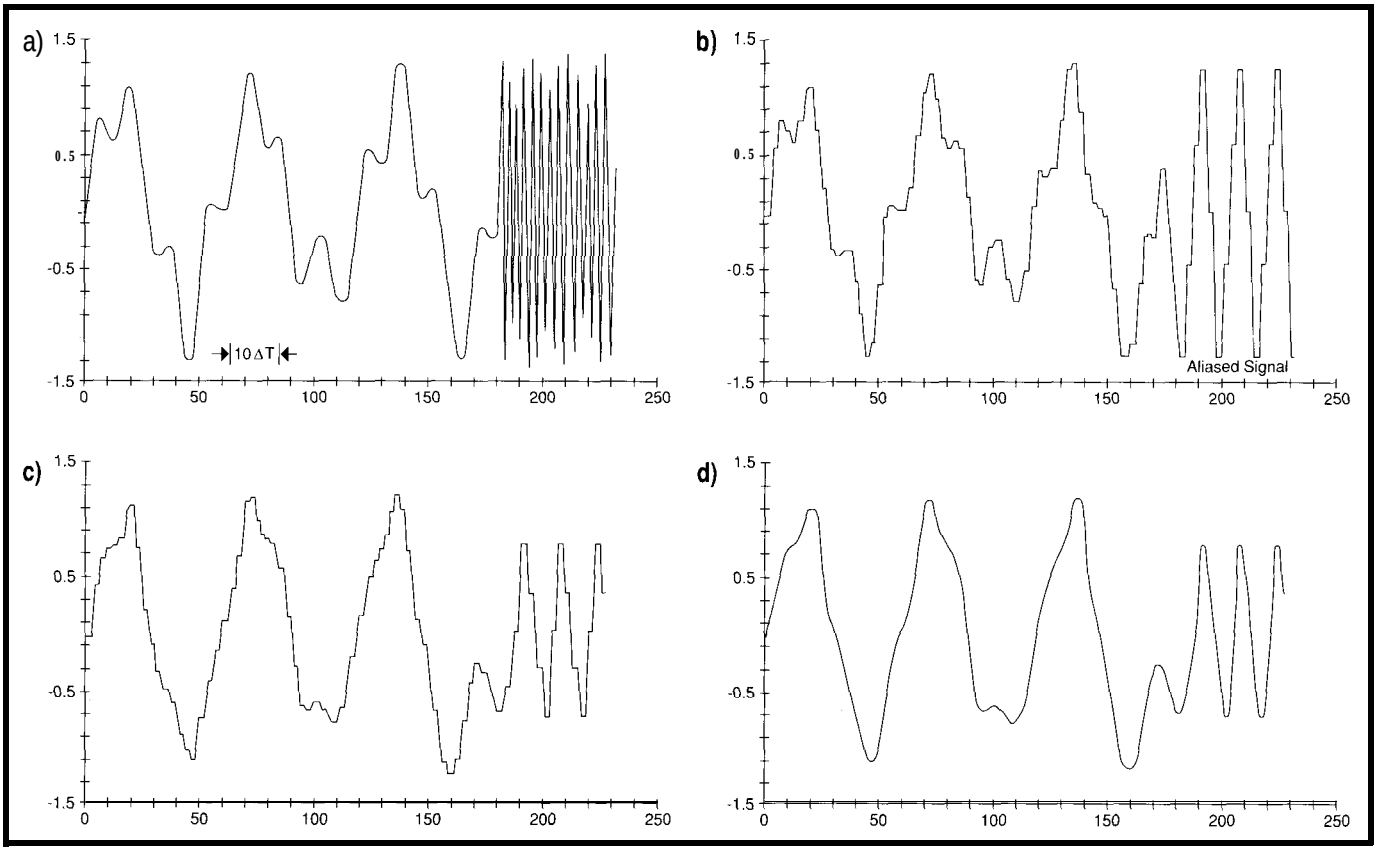


Figure 3—*(a) The* analog *input signal contains both* high- *and low-frequency components. (b) When digitized, the high-frequency signal is lost to aliasing. (c) The digital output of the filter shows that only the lower frequencies have been passed. (d) The final analog output shows the filtered signal.*

various features of the chip allowed me to get a more efficient algorithm.

$$y[n\Delta T] = x\left[\left(n-\frac{N-1}{2}\right)\Delta T\right] \times coeff\left(\frac{N-1}{2}\right) + \sum_{i=0}^{(N-1)/2} \{x\,[(n-i)\,\Delta T] + x\,[(n-N+i)\,AT]) \times coeff\,[i]$$

$n \geq N$

The first part of Listing 1 shows the full C code for an on-line FIR filter. The second part is a direct translation of this code into Am29050 assembler. The program starts by initializing pointers that are used to bring in data and filter coefficients from memory. Then follows a loop to bring in the data and coefficients from external memory to implement the filter.

One difference between on- and off-line filtering is the need to service the A/D and D/A converters. Less obvious is that the real-time loop will be slower because of the overhead of the circular buffer, since the data must remain within the bounds of a fixed memory array.

If I assume that the CHECKADJUST macro takes four instructions, the real-time FIR filter takes $(8N+15)$ instructions. This is a considerable period of time to perform $2N$ floating-point operations. The program is very inefficient, especially if I account for the possibility of wait states and the problem of keeping the pipelines full. To improve the efficiency, I wrote the code again, taking into account the architectural features of the Am29050.

The Am29050 has 192 registers which can be used as a cache. Since the filter coefficients are unchanging, it would make sense to store these in the on-board registers. The chip also has separate data and instruction buses so I can load data during instruction execution.

The program causes frequent pipeline stalls. This is particularly true for floating-point operations where the pipeline is six stages deep. By rearranging the code, I can keep the pipelines full and gain considerable speed.

Each branch will cause the instruction pipeline to be flushed—which causes it to stall. Activating the on-board branch target cache (BTC) stores the instructions following the most recent branches and reduces instruction pipeline stalls.

A major problem is the overhead associated with the circular buffer that occurs during address calculations. The memory management unit can perform this process in parallel with other operations, giving a significant performance improvement. I can also gain speed with the ASS E RT instruction, which does single-cycle branch-and-test during address pointer checks.

DATA CACHE FULLY ON-BOARD THE INTERNAL AM29050 REGISTERS

If the number of taps is small $(N \leq 102)$, it is possible to store all data and coefficients on-board. The filter coefficients are stored using global registers (gr64–gr114). The data values are stored in local registers (lr0–lr101). The more-optimized code for implementing a real-time FIR filter for this situation is shown in Listing 2. Data values are placed in local register window. This avoids the overhead of a circular buffer, because the local registers in conjunction with the

Listing 1—*The* first *part shows the C implementation of an FIR filter while the second part shows the same* algorithm *written* in Am29050 assembler. *This* direct **approach** causes stalls in the *FPU,* data, and instruction *pipelines and results in a grossly inefficient program.*

```
   .cputype 29050

; ASSUME NUMTAPS IS ODD

;float  xarray[BUFFSIZE], coeffarray[1 + NUMTAPS / 2];
; float *xstart, *xrecentpt. *xancientpt, *ypt, *coeffpt
; float *buffstart, *buffend;
; float yn;
;
; xstart = xarray + NUMTAPS 1;        /*avoid range problems*/
; buffstart = xarray;
; buffend = xarray + BUFFSIZE 1

;  for (;;){
     yn = 0;
     coeffpt = coeffarray;             /*establish pointers*/
     xrecentpt = xstart:
     xancientpt = xrecentpt NUMTAPS 1;
     if (xancientpt < buffstart) xancientpt += BUFFSIZE:
     *xstart = (float) ReadA2D();
     for (i = 0; i < NUMTAPS / 2; i++)
     {/* do FIR */
        yn += *xrecentpt- * coeffpt; /*worry about wrap*/
        if (xrecentpt < buffstart) xrecentpt = buffend;
        yn += *xancientpt++ * *coeffpt++:
        if (xancientpt > buffend) xancientpt = buffstart;
     }
     yn += *recentpt * *coeffpt             /*centre tap*/
     WriteD2A( (int) yn);
     xstart++;
     if (xstart > buffend) xstart = buffstart:
; }

; Array info
   .set     FPACC. 0                  ; which FP Acc to use

   SETVALUES                          ; initialize values
LOOP:
   fmac     AB_PLUS_0, FPACC, ZERO, ZERO ; yn = 0
   or       COEFFPT, COEFFARRAY, 0    ; coeffpt = coeffarray
   or       XRECENTPT, XSTART, 0      ; xrecentpt = xstart

   LCONST   XANCIENTPT. NUMTAPS

   sub      XANCIENTPT, XANCIENTPT, 1
   srl      XANCIENTPT, XANCIENTPT, 2    ; turn into a byte offset
   sub      XANCIENTPT, XRECENTPT, XANCIENTPT

   CHECKADJUST XANCIENTPT
   load     0, 0, XVALUE, A2D        ; *xstart = (float) ReadA2D()
   convert XVALUE, XVALUE, SIGNEDDINT, NEAREST. SING_FP, INT
   store    0, 0, XVALUE, XSTART

   .rep     (NUMTAPS >> 1)
   load     0, 0. COEFF, COEFFPT      ; coeff = *coeffpt++
   add      COEFFPT, COEFFPT. 4

   load     0, 0, XVALUE, XRECENTPT ; yn += *xrecentpt- * coeff

   sub      XRECENTPT. XRECENTPT. 4
   fmac     AB_PLUS_ACC, FPACC, XVALUE, COEFF

   CHECKADJUST XRECENTPT

   load     0, 0, XVALUE, XANCIENTPT  ; yn += *xancientpt++ * coeff
```

(continued)

Listing 1—*continued*

```
   add      XANCIENTPT, XANCIENTPT, 4
   fmac     AB_PLUS_ACC, FPACC, XVALUE, COEFF

   CHECKADJUST XANCIENTPT

   .endr

   load     0, 0, COEFF, COEFFPT          ; coeff = *coeffpt
   load     0, 0, XVALUE, XRECENTPT       ; yn += *xrecentpt * coeff

   fmac     AB_PLUS_ACC, FPACC, XVALUE, COEFF ; /* Centre tap */
   mfacc    YVALUE, SING_FP, FPACC        ; WriteD2A( (int) yn )

   convert  YVALUE, YVALUE, SIGNED_INT, NEAREST, INT, SING_FP
   store    0, 0, YVALUE, D2A
   add      XSTART, XSTART, 4             ; xstart++;
   CHECKADJUST XSTART

   jmp      LOOP
   nop                                    ; delay slot
```

Listing 2—*On-line FIR filter code using the Am29050 on-board registers as a data cache. Fully customizing the FPU operations leads to a three-fold greater speed.*

```
   .cputype 29050
   .set     NUMTAPS, 103
   .set     NUMCOEFFS, 52
   .reg     H0, gr64              ; coeff CACHE  start
   .reg     HCENTRE, gr115        ; centre tap coeff
   .reg     X0, lr102             ; 0 delay sample
   .reg     XCENTRE, lr51         ; centre tap value
   .reg     XLAST, lr0            ; max delay sample
   .set     XVALUE, X0            ; working variables
   .set     A2D, gr127            ; A/D address stored here
   .set     D2A, gr126            ; D/A address here
   .set     YVALUE, gr125
   .set     COEFFARRAY, gr124     ; coeff[] address
   .set     FPACC, 0              ; which Acc to use
   SETREGISTERS                   ; initialize registers
   mtsrim   cr, (NUMCOEFFS - 1)
   loadm    0, 0, H0, COEFFARRAY  ; fetch coeffs
LOOP:
   load     0, 0, XVALUE, A2D     ; get value from A/D
   convert  XVALUE, XVALUE, SIGNED_INT, NEAREST, SING_FP, INT

   fmac     AB_PLUS_0, FPACC, ZERO, ZERO      ; set sum = 0
   fmac     AB_PLUS_ACC, FPACC, X0, H0        ; sum += X0 x H0
   fmac     AB_PLUS_ACC, FPACC, XLAST, H0     ; sum += XLAST x H0
   .set     N, 0
   .rep     (NUMCOEFFS - 2)
   .set     N, N+1                ; sum += (Xn + X(LAST-n]) * Hn
   fmac     AB_PLUS_ACC, FPACC, %%(&X0 - N), %%(&H0 + N)
   fmac     AB_PLUS_ACC, FPACC, %%(&XLAST + N), %%(&H0 + N)
   .endr
   fmac     AB_PLUS_ACC, FPACC, XCENTRE, HCENTRE ; centre tap val.

   mfacc    YVALUE, SING_FP, FPACC ; get sum out of FP unit
   convert  YVALUE, YVALUE, SIGNED_INT, NEAREST, SING_FP
   store    0, 0, YVALUE, D2A     ; write (int) YVALUE

   sub      gr1, gr1, 4           ; adjust circular buffer
   jmpfdec  LENGTH, LOOP
   nop                            ; can fill later
```

Am29050's *register stack pointer* (grl) automatically form a circular buffer.

Although there are now only one eighth of the original number of instructions, the speed is only a factor of two faster. This is because removing the circular buffer exposed inefficiencies in how I used the FPU. By keeping the FPU pipeline full, this code could run in $(N+15)$ instructions and take only $(N+18)$ cycles. This is a 800% speed advantage over the first method of implementing the filter.

DATA CACHE PARTIALLY ON-BOARD THE INTERNAL AM29050 REGISTERS

When the number of taps is large $(N>103)$, it is not possible to store data and filter coefficients on chip. Therefore, memory accesses are required. Next I show how to overcome many memory access problems related to Listing 1 and some hidden problems in using on-board registers and the FPU as related to Listing 2.

Considerable overhead is associated with updating data pointers. The Am29050 has a short machine cycle, and does not implement an autoincrementing address register. Address calculations can be performed using hardware that operates in parallel with other facilities. This capability is used with the load multiple (LOADM) instruction, which has the capability of transferring a memory block to a register block. This overlaps an implicit autoincrement calculation with data fetches.

The READBLOCK macro reads in NUMBER data values from a memory block pointed to by ADDRESSREGISTER into registers starting at DATAREGISTER as shown in Listing 3.

The Am29050 is a 40-MHz chip and it is expensive to implement zero-wait-state memory for this processor speed. Instead, consider burst memory-where after an initial wait from the first read request, data is available each cycle. The macro READBLOCK uses the LOADM instruction, so it makes efficient use of burst mode memory. The *instruction prefetch buffer* is designed to allow burst memory. This reduces memory speed requirements and design cost.

Listing 3—*READBLOCK read; NUMBER data values* from a memory *block pointed to by ADDRESSREGISTER* into *registers starting at DATAREGISTER. The DoFIR* macro multiplies NUMBER data *values stored at register DSTART with the* coefficients stored at *register CSTART.*

```
macro   READBLOCK, DATAREGISTER, ADDRESSREGISTER, NUMBER
mtrsim cr,  (NUMBER  1)                    ; load address counter

; Round them up, move them in
loadm   0, 0, DATAREGISTER, ADDRESSREGISTER

.endm

macro   DoFIR, DSTART, CSTART, NUMBER, DIRECTION
.set  N,  -1
.set    FPACC,  -1
.rep    NUMBER
.set    N, N+1
.set    FPACC, ((FPACC+1) % 4)             ; which FP Accumulator

FMAC AB_PLUS_ACC, FPACC, %%(&DSTART-N), %%(&CSTART+DIRECTION*N)

.endr
.endm
```

KEEPING THE FLOATING POINT PIPELINE FULL

After modifying the code to take advantage of on-board registers or burst memory, you would expect the filter speed to improve significantly, but it doesn't. The inefficiency now lies in the use of the FPU shown in Figure 4. The floating-point multiply-and-accumulate FMAC instruction makes use of the majority of the FPU resources. Every FMAC instructions takes 6 cycles. However, the pipeline implies a new FMAC instruction can

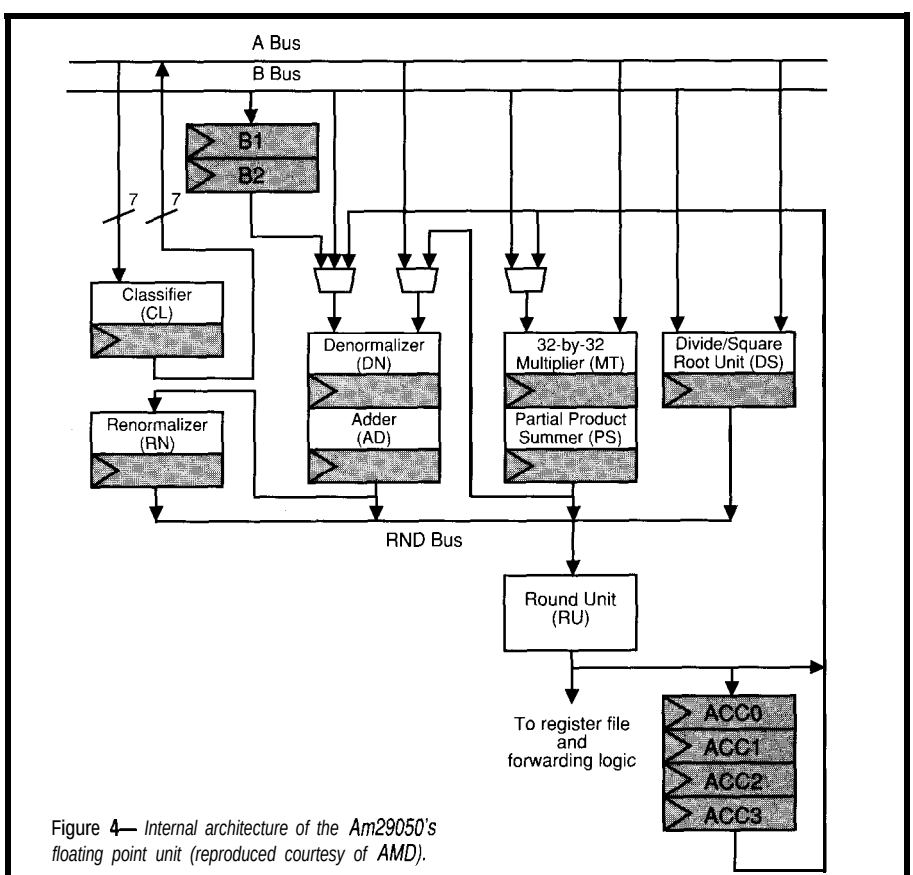


Figure 4— *Internal architecture of the Am29050's floating point unit (reproduced courtesy of AMD).*

start (and complete) every 25 ns. I am not getting that kind of performance! So where is the problem?

In checking the data books, I find that although an FMAC instruction can start every cycle, it cannot complete until all data values are available. Since four FMAC instructions all make use of the single floating-point accumulator at pipeline stage 3, they stall at that point until the previous FMAC instruction completes. This problem means that each FMAC stalls for three cycles before it proceeds. This hidden inefficiency did not show up in the original code. The memory accesses to get data and coefficients were interleaved with the FMAC operations so each FMAC found its operands. Removing the memory accesses that occurred in parallel with FPU operation exposed this inefficiency.

The Am29050 designers were aware of this problem and provided additional floating-point accumulators to overcome stalls. Rewriting the code to use all four accumulators fixes the problem.

The macro DoFIR, shown in Listing 3, multiplies NUMBER data values stored at register DSTART with the coefficients stored at register CSTART. It takes all four accumulators into account to calculate the partial sum and the data values are stored at the highest number's register. The variable DIRECTION takes into account the coefficients for filter tap N-rissameasfortap r. The DoFIR and READBLOCK macros can be used in on- and off-line filters.

IMPLEMENTATION OF A NO-OVERHEAD CIRCULAR BUFFER ON A RISC CHIP

A fundamental deficiency on RISC chips is the small number of addressing modes in the instruction set. Implementing circular buffers would appear to incur considerable overhead.

A simple array can be turned into a circular buffer by using pointers and pointer comparison. In simple pseudocode, the ADJUST macro used to adjust the data block pointers becomes:

```
.macro ADJUST, POINTER
POINTER++
IF (POINTER>x[N-1])
   POINTER=x[0]
.endm
```

The autoincrementing LOADM instruction can't be used to speed up register loading with on-line circular buffers because during the instruction, no range checking would be performed. Apparently the registers must be loaded individually using the LOAD instruction and the ADJUST macro executed at every data fetch in the loop. This is considerable overhead.

The memory management unit and *translation look ahead buffer,* (TLB), hardware can be programmed to perform the pointer checking in parallel with other instructions using low overhead software tricks. A circular buffer can be implemented by using the *virtual addressing mode* during the loop. The circular buffer is obtained by mapping several virtual addresses to a single physical address.

The memory management unit can block memory pages from 1K to 4K, so it's possible to generate addresses in modulos that are multiples of 256 words. Note that my circular buffer is 512 words. Only 193 locations are used at any time.

Setting the TLB registers is simple. They can be initialized by masking the required virtual and physical addresses. The macro SETTBL2K (shown in Listing 4) maps the address VIRTUAL into the address PHYSICAL and sets the TLB registers. This macro does not turn on the virtual addressing mode since it may be invoked many times before all the required TLB registers have been set up. The new ADJUST macro needs no error checking and becomes:

```
.macro ADJUST, REGISTER
add REGISTER, REGISTER, 4
.endm
```

This approach to the circular buffer means the real-time FIR code looks like Listing 5 and requires $(2N+3$ 1) instructions. This is a 400% speed improvement over the first implementation. I still have to use:

```
.macro CHECKADJUST, REGISTER
.set CIRCBUFFTRAP,
        CIRCBUFFERTRAPNUM
asleu CIRCBUFFTRAP, REGISTER,
        CIRCBUFFEREND
.endm
```

once each loop to ensure that the pointers remain in the circular buffer, but this is not significant overhead. The ASLEU instruction performs the pointer comparison in a single cycle. If the REGISTER address gets too high, an *assert trap* is caused. The overhead associated with this trap handling is about seven cycles every 5 I2 times through the loop. This is less than the three cycles every loop for using a compare (CPLEU), jump (JUMPT), and delay slot combination of instructions.

EVALUATION OF THE AM29050 AS A CHIP FOR FIR OPERATION

The Am29050 has many features found in DSP chips. The register stack can be used for data and coefficient storage. It has a separate instruction bus so data and instruction fetches do not compete. It has a number of ALUs that operate in parallel, and the FPU operates in a single cycle. An advantage of the Am29050 is that it is a RISC chip and the internal architecture is exposed so I can maximize the throughput of the pipelines.

What, then, is missing from the Am29050 as a "perfect" DSP chip? Several things come to mind from examining the FIR algorithm.

The Am29050 has a limited amount of on-board RAM memory (192 registers); it is on the small side when compared with the newer DSP chips, but is still large enough to be useful. The problem is reloadability, being defined as the ability to reload one part of the register file from external memory while processing using another part. Currently, while floating-point operations are performed, the external address and data paths are inactive. If these paths could be used in parallel, the speed would increase. Since the Am29050 is a RISC chip, developing complex instructions for parallel operations goes against its basic design. There is, however, one important instruction where this

Listing 4—*The SETTBL2K* macro is used to implement 2K *circular buffer operations with zero overhead.*

```
  .macro SETTLB2K, PHYSICAL, VIRTUAL

      .set      VTAGMASK, lr2              ; set registers
      .set      TLBLINEMASK, lr3
      .set      RPNMASK, lr3
      .set      VE, lr5
      .set      PROTECTION, lr6
      .set      MMUPS, lr7
      .set      TLBLINE, lr8
      .set      TLBWORDO, lr9
      .set      TLBWORDl. lr10
      .set      TEMP, lr11

      LCONST    VTAGMASK, 0xFFFF0000       ; see manual for defins.

      LCONST    TLBLINEMASK, 0xF800        ; for 2K pages operation

      LCONST    RPNMASK. 0xFFFFF800
      LCONST    MMUPS, 0x100
      LCONST    VE, 0x4000                 ; valid entry bit
      LCONST    PROTECTION, 0x380          ; supervisor mode

; Find out which line in TLB registers are needed
      and       TLBLINE, VIRTUAL, TLBLINEMASK
      srl       TLBLINE, TLBLINE, 10       ; 2K mode

; See if this TLB set 0 is already in use
      mfsr      TEMP, TEMPLINE
      and       TEMP, TEMP, VE
      cpeq      TEMP, TEMP, VE             ; is VE bit set?
      jmpf      .+oxc                      ; skip to and
      nop
      add       TLBLINE, TLBLINE, 64       ; change to other set

; Build the TLB words and move them in
      and       TLBWORDO, VIRTUAL, VTAGMASK
      or        TLBWORDO, TLBWORDO, VE
      or        TLBWORDO, TLBWORDO, PROTECTION
      mtsr      TLBLINE, TLBWORDO

      add       TLBLINE, TLBLINE, 1
      and       TLBWORDl, PHYSICAL, RPNMASK
      mtsr      TLBLINE, TLBWORDl

      mtsr      mmu, MMUPS                 ; set mmu page size

  .endm
```

Listing 5—*Am29050 processor code for the implementation of a real-time 193-tap FIR filter using data cache, full FP pipeline coding, hardware modulo arithmetic, and simultaneous FP and integer operation coding strategies.*

```
   .include "startup.s"

.macro ADJUST, REGISTER

   NOTDEFINED

.endm

   .set       NUMTAPS, 193
   .set       NUMCOEFSS, 97
   .set       FILTERCOEFF, Ox18000
```

(continued)

Listing 5—*continued*

```
    .set        BUFFSIZE, 0x400
    .set        PHYSICALBUFFER, 0x19000 ; physical address
    .set        CIRCULARBUFFER, 0xFF9000  ; virtual address
    .set        CIRCBUFFERTRAPNUM, 65  ; user dependent

    .set        A2DADDRESS, 0xFFF40000
    .set        D2AADDRESS, 0xFFF40004
    .set        TEMP, gr83 code

    .set        A2D, gr84               ; A/D address
    .set        D2A, gr85               ; D/A address
    .set        CIRCBUFFEREND, gr86

; Set TLB for virtual address mapping
;(NB gr64->gr67 used as temps in use later)
    .set        CIRCBUFFPT, gr64        ;circ buffer virtual addr

    .set        PHYSICALPT, gr65        ;circ buffer physical addr

    .set        TWOK, gr66              ; 2K page
    .set        TEMP1, gr67             ; used to set TRAP addr

    LADDR       A2D, A2DADDRESS
    LADDR       D2A, DZAADDRESS
    SETTLB2K    A2D, A2D                ; physical = virtual
    SETTLB2K    D2A, D2A                ; physical = virtual

    LCONST      TWOK, 0x800             ; 2K blocks
    LADDR       PHYSICALPT.  PHYSICALBUFFER
```

(continued)

parallel operation appears practical—the load multiple (LOADM) instruction.

When I originally wrote this article, I had the following suggestion: If this had simply been a software problem, then the programmer/compiler would have been warned about reading data before it was ready and avoided the conflict. RISC chips and the compiler are supposed to interact for optimium code and perhaps a version of the Am29050 chip could be released for DSP users without the data dependency between the LOADM and the FP instructions being hardwire checked and simply relied on the compiler/assembler to check the data dependency.

However, since that time I have become older and wiser and have had experience with a chip without the hardware checks-the Intel i860. A fast chip it might be, but to get that maximum performance requires very detailed knowledge of the chip architecture because of the missing interlocks. My attitude now is hardware checks (scorecarding) are an

essential feature of a processor as it allows the programmer to get "excellent" performance without becoming intimate with the architecture.

Another problem is that currently, to add the values of two floating-point accumulators used in multiple FMAC instructions, the register values must be brought out of the FPU into local registers (M FACC) and then added. The lack of such an add instruction increases the length of the 193-tap FIR loop by about 3 cycles in 400, so it is relatively unimportant. However, for smaller DSP loops such as in the IIR or FFT, it might be more relevant.

CONCLUSION

In this article I examined the application and theory behind the implementation of high-speed FIR filters on the Am29050. I also illustrated that the Am29050 processor had many architectural features commonly found in DSP chips.

One major DSP feature missing is the ability to allow external block data memory access to occur in parallel with floating-point operations. This would allow the on-board data cache to be refilled from external memory with minimum overhead.

In addition to being a high-speed general-purpose processor, the Am29050 is no slouch as a DSP chip. Will I see this processor in industrial DSP applications? The performance is already there and the floating-point operation allows for fast development time. There is currently a problem associated with power consumption and chip cost. The newly released Am29240, based on the same 29k processor core as the Am29050, offers some interesting possibilities. It is an integer chip, has on-board instruction and data cache, and a single-cycle pipelined hardware multiplier. I am currently exploring using this chip for spectral analysis. ❑

Michael Smith is a professor of Electrical and Computer Engineering *at the University of Calgary, Canada, where he teaches courses on Computer Graphics, Comparative Processor Architecture, and Systematic Programming Techniques.*

Listing 5—*continued*

```
    LADDR       CIRCBUFFPT,  CIRCULARBUFFER
    SETTLBZK    PHYSICALPT, CIRCBUFFPT ; circ buffer
    SUB         TEMP, CIRCBUFFPT. TWOK
    SETTLBPK  PHYSICALPT,  TEMP         ; block below
    ADD         TEMP, CIRCBUFFPT. TWOK
    SETTLBZK    PHYSICALPT, TEMP        ; block above
    SUB         CIRCBUFFEREND, TEMP, 4 ; used in pointer checks

    ADD         TEMP. TEMP, TWOK
    SETTLBZK    PHYSICALPT, TEMP        ; second block above

    SETASSERTTRAP  CIRCBUFFERTRAPNUM,  circbufftrap

    LADDR       XOPT. (CIRCBUFFPT + 4 * NUMTAPS)
    sub         X24PT, XOPT, (DATABLOCKSIZE * 4)
    sub         X48PT, X24PT, (DATABLOCKSIZE * 4)
    sub         X72PT, X48PT, (DATABLOCKSIZE * 4)
    sub         XCENTREPT, X72PT, (DATABLOCKSIZE * 4)
    sub         X97PT, XCENTREPT, 4
    sub         X121PT, X97PT, (DATABLOCKSIZE * 4)
    sub         X145PT, X121PT, (DATABLOCKSIZE * 4)
    sub         X169PT, X145PT, (DATABLOCKSIZE * 4)

    LADDR       TEMP. FILTERCOEFF
    mtsrim      cr, (NUMCOEFFS 1)
    loadm       0, 0. h0, TEMP          ; move in the filter coeffs.

    mtsrim      cps, (PI | SM   DI)     ; turn on virtual data mode

LOOP:
    READBLOCK X0, XOPT, DATABLOCKSIZE; load in data
    load        0, 0, XVALUE, A2D       ; get value from A/D
    convert XVALUE,XVALUE,SIGNED_INT,NEAREST,SING_FP,INT

    ; Clear the accumulators and avoid blocking x0

    fmac        AB_PLUS_0, SUM0, X1. H1; SUM[0] = X1 x H1
    fmac        AB_PLUS_0, SUM1, X2, H2: SUM[1] = X2 x H2
    fmac        AB_PLUS_0, SUM2, X3. H3: SUM[2] = X3 x H3
    fmac        AB_PLUS_0, SUM3, X0, H0: SUM131 = X0 x H0

    DOFIR       X4,  H4,  (DATABLOCKSIZE  4), +1 ;X4xH4 -> X23, HZ3

    store       0, 0, XVALUE, XOPT      ; moved fp store-don't block

    ADJUST      XOPT                    ; adjust buffer pt

    READBLOCK   X24.  X24PT, DATABLOCKSIZE       ; get next block
    DOFIR       x24   H24, DATABLOCKSIZE, +1     ;Update X24PT moved
    READBLOCK X48   X48PT, OATABLOCKSIZE         ;start this
    DOFIR       X48   H48, DATABLOCKSIZE, +1     ;Update X48PT moved
    READBLOCK X72   X72PT, DATABLOCKSIZE
    DOFIR       x72   H72, DATABLOCKSIZE, +1

    READBLOCK X97, X97PT, DATABLOCKSIZE
    DOFIR X97, H97, DATABLOCKSIZE, -1 ;coeffs symmetric

    READBLOCK X120, X120PT, DATABLOCKSIZE
    DOFIR       X121, H121, DATABLOCKSIZE, -1
    READBLOCK X145, X145PT, DATABLOCKSIZE
    DOFIR       X145, H145, DATABLOCKSIZE, -1
    READBLOCK X169, X169PT, DATABLOCKSIZE
    DOFIR       X169, H169, DATABLOCKSIZE, -1

    load        0,0 XCENTRE,  XCENTREPT
    fmac        AB_PLUS_0, SUM0, XCENTRE,  HCENTRE      ; centre tap
```

(continued)

Listing *5-continued*

```
    mfacc     TEMP1,  DOUBLE-FP,  SUM1  ; templ  = sum1
    mfacc     TEMPP,  DOUBLE_FP,  SUM2
    mfacc     TEMP3,  DOUBLE_FP,  SUM3
    mfacc     TEMPO,  DOUBLE_FP, SUMO

    dadd      TEMP1,  TEMP1.  TEMP2     ; templ  += temp2

    ADJUST    X24PT                     ;so temp0  ready
    dadd      TEMPO.  TEMPO,  TEMP3     ; temp0  +=  temp3
    ADJUST    X48PT                     ;so temp0  ready
    ADJUST    X72PT
    dadd      TEMPO,  TEMPO,  TEMP1     ; temp0  +=  templ

    ADJUST    X97PT
    ADJUST    X121PT                    ; get temp0  ready
    convert   Yn,  TEMPO,  SIGNED_INT,  NEAREST,  INT,  DOUBLE_FP
                                        ; yn  = (int) temp0
    ADJUST    X145PT
    ADJUST    X169PT
    ADJUST    XCENTREPT

    ; check and adjust the pointers for the circular buffer
    CHECKADJUST  X169PT                 ; finish the convert

    jmpfdec   COUNTER,  LOOP
    store     0,  0,  Yn,  D2A          ; output  Yn

circbufftrap:
    CIRCBUFFTRAPHANDLER
```

SOFTWARE

Software for this article is available from the Circuit Cellar BBS and on Software On Disk for this issue. Please see the end of "ConnecTime" in this issue for downloading and ordering information.

REFERENCES

T.J. Terell, ***Introduction to Digital Filters,*** MacMillan Press, 1983.

Am29050 Microprocessors User's Manual, AMD, 1991.

For more information on the Am29050 processor DSP library, contact the AMD 29k hotline: (800) 2929-AMD (U.S.], (800}531-5202 (Canada).

I R S

401 Very Useful
402 Moderately Useful
403 Not Useful

Using Your PC for Function Analysis and Control

FEATURE ARTICLE

David S. Birkett

With the addition of a bit of hardware and software, there is a test bench hidden inside your PC. Find out how to control chip-level function generators and analyze incoming waveforms, all with your PC.

Computer-based instrumentation is becoming more and more common. A diverse new breed of instruments that have varying levels of computer control are starting to appear in the marketplace. On one hand, there are the software-based simulations of "hardware experiments." These "software labs" give you the ability to experiment with and observe the operation of experimental circuits without having to actually wire up real components on breadboards. On the other hand, there is a wide range of computer-controlled versions of the traditional mix of bench gear. At this end of the spectrum are devices like computer-based logic analyzers, power supplies, frequency counters, oscilloscopes, and so forth. A related trend is to embed more intelligence into test equipment and provide it with a command interface so it can be controlled by, or send data to, a separate computer system. These trends will continue and, in all likelihood, will change the nature of testing and test equipment. This will lead to new methods of circuit design and testing.

I have coupled my PC with a simple hardware/software system, and

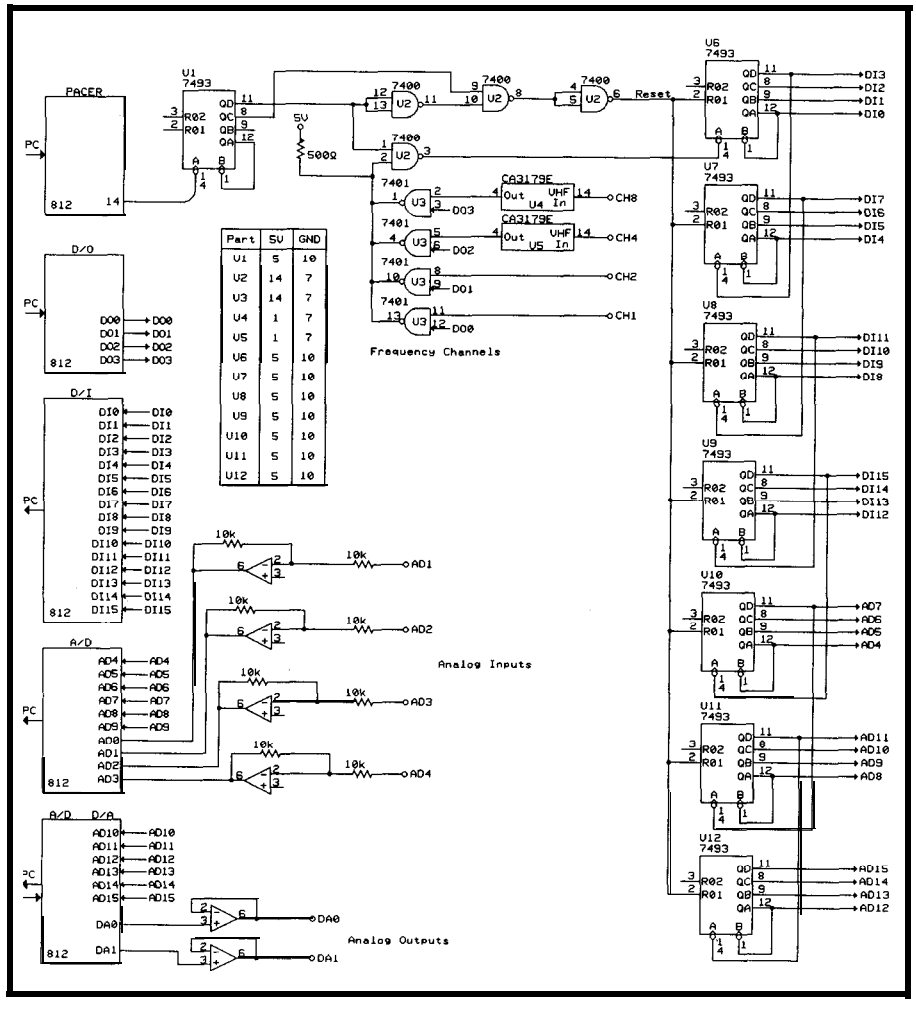


Figure I--Based on an *off-the-she/f PC I/O expansion card, the function analyzer adds the basics necessary for measuring and controlling real-world signals.*

in doing so, I have created an instrument that has great potential in laboratory control and measurement applications. I call this new instrument a *function analyzer*. In essence, it allows any voltage-controlled oscillator (VCO) to be controlled by keyboard. Now I can control a variety of inexpensive IC-based signal sources. This control interface transforms them into precise, versatile frequency synthesizers. I gave the function analyzer its name because it operates on mathematical principles. It uses an IBM-compatible PC equipped with a multifunction I/O card that has been enhanced with an elementary counter.

```
COUNT          2219500     RESOLUTION  -64
GATE(SECS)     2
D/O            8
FREQ           1110085     FREQ*64     7.104546E+07
STABILITY      -47
AD0            4.358108
AD1            7.030303
AD2            4.98646
AD3            4.967082

READ COUNT AGAIN? (Y/N) Y
```

Figure 2—*Single-counter listing (item 6) includes channel number, raw count, gate period and resolution, frequency = count/gate period, frequency • 64 (for use with prescaled channel), frequency stability with respect to previous reading, and four analog voltages.*

HARDWARE

Figure 1 shows the schematic of the computer interface and some of the external hardware that is required by the function analyzer. Notice that there are five ports available on the multifunction I/O card. This interface provides the user with 16 digital inputs, 16 digital outputs, 16 analog inputs, 2 analog outputs, and a programmable pacer derived by dividing a 4-MHz crystal.

I created the function analyzer by designing a circuit that could attach to the PC through a PLC812PG card from B&C Microsystems. I decided to do this because this interface card is easily installed and, through its driver and utilities, offers a programmable interface that can be enhanced for just about any purpose. Another reason I chose to work with this card is that its hardware interface is well documented and it also includes a fair set of sample programs written in BASIC that illustrate how to take advantage of various facets of the card. You can use this information, like I did, to come up with a wide variety of custom I/O applications.

In my application, seven 4-bit binary counters (U6–U12) are used to make a 28-bit frequency counter. U1 and U2 cyclically gate, freeze, and reset the counter chain at a rate that is set by the programmable pacer. The counter software, shown in Listing 1, continuously reads 16 digital and 12 analog inputs. These readings are used to recognize when a valid frozen count has occurred which, in turn, is used to compute a frequency referenced to the pacer crystal.

The design of this counter makes the system very versatile. The independently cyclical operation of the hardware and software makes implementing a variety of counter-based experiments simple, flexible, and reliable. The system is equipped with a quartz timebase and boasts a resolution of 1 part in 2^{28} (1 in 10^8). Its gate is continuously adjustable from the keyboard, and gate times can vary from 0.1 second to several minutes. It has four input channels, and channel selection is also selectable from the keyboard. The CA3 179E prescaler chips give the counter a wide frequency range. This range extends from a few hertz at the low end to gigahertz at the top end of the range. These frequency measurements are available for processing and display by the PC.

Figure 2 shows the PC's output from a single reading. The data represented by Figure 2, in addition to frequency information, presents simultaneous measurements from the analog portion of the circuit. These are used to monitor analog outputs that control VCOs or to record other corollary voltages. All of the analog lines are buffered by operational amplifiers in order to protect the I/O card. As is usual in PC-based instrumentation, the analog inputs are calibrated in software against external voltage standards. Software calibration gives the function analyzer's voltage readings a certain degree of absolute accuracy and makes implementation of the signal buffers trivial. This

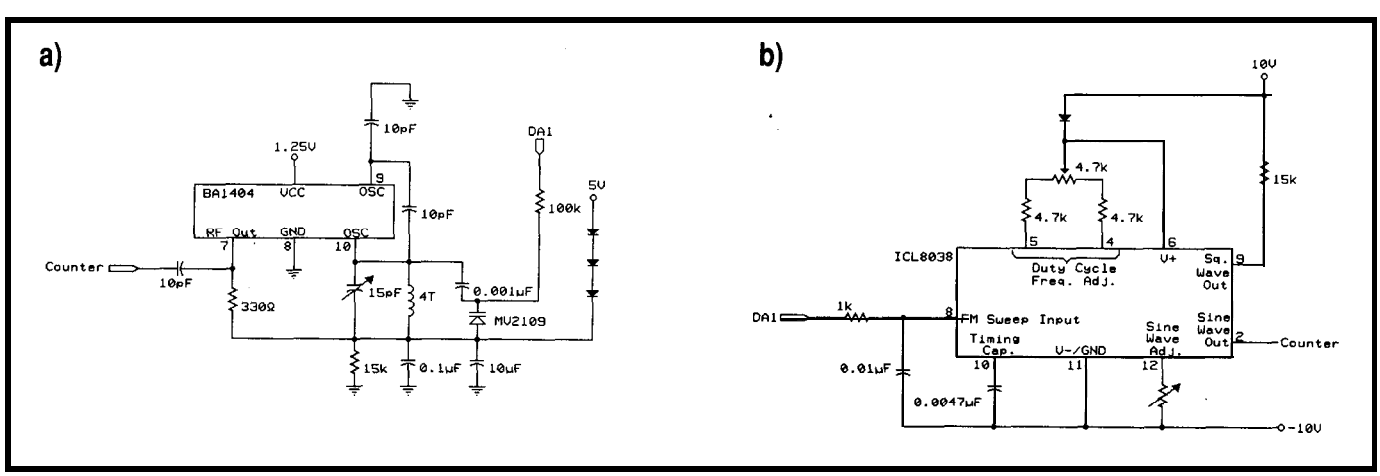


Figure 3—*Two examples of function generator chips that can be controlled by the function analyzer include a) the BA 1404 stereo FM transmitter chip and b) the ICL8038 precision waveform generator.*

accuracy is assured as long as the buffers are stable and linear. The gain of the individual buffers may be set in accordance with your specific hardware considerations.

FREQUENCY SYNTHESIS

Figure 2 may be considered an example of "direct" operation of the function analyzer. In direct operation, a control parameter is used to set an analog voltage level, which is then applied to a VCO. Next, various calibrated frequency and voltage measurements are recorded. These show the system's response to the settings of the control parameter. Figure 3a shows a BA1404 FM transmitter chip which is tuned with an external varactor to operate at approximately 60 MHz (Figure 3b shows another potential application using an ICL8038 waveform generator chip). Software can be easily written which will step the voltage applied to the varactor through a range of values, and thereby change the frequency of the chip. The table in Figure 4 lists a sequence of D/A settings (this is the control parameter in this experiment and its settings are shown in column 1). You can read the frequency measured by the counter during each step by looking at column 2. Column 3 lists the varactor's actual control voltage as measured by a calibrated ADC. The measurements of Figure 4 completely describe the operation of the VCO over a range of interest.

Figure 4 shows that the frequency and voltage of this system are functions of the control parameter. These

```
STEP DA1, ESTABLISH Y1, THE CALIBRATION MATRIX

ENTER FIRST DA1? 2200

ENTER LAST DA1? 4000

ENTER INCREMENT? 300

XI = 500 X2=500 DA0=0 D/O=8

  DA        FREQ.        AD     AD    DY/DX   ACC   STAB   MAG   TEST
2200.0   58204610.0     2.67   4.30   0.00     44    -14     0   0.0000
2500.0   60895620.0     3.03   4.89   0.00     43    -14     0   0.0000
2800.0   64898110.0     3.39   5.47   0.00     43    -12     0   0.0000
3100.0   67698180.0     3.75   6.06   0.00     43    -14     0   0.0000
3400.0   69847040.0     4.12   6.65   0.00     43    -15     0   0.0000
3700.0   71594880.0     4.48   7.23   0.00     42    -17     0   0.0000
4000.0   73068800.0     4.84   7.81   0.00     42    -17     0   0.0000

FREQ  CHANNEL=        8
```

Figure 4—*Direct control of the BA* 1404 via external varactor tuning. The PC *DAC control parameter is stepped from* 2200 to 4000 *in seven steps and the VCO frequency and varactor voltage are measured.*

Listing *I-Software for acquisition of data from counter chain of Figure 1. Lines 420–480 provide initialization, lines 490–680 repeated/y read the counter, and line 690 recognizes a frozen nonzero count and jumps to the end of the program.*

```
420     FUN%=22
430     CALL PCL1812 (FUN%,DAT%(0),ER%)
440     DI=DAT%(0) + DAT%(1) * 256
460     S = INT(PITCH * MUTE * 2 ^(J/12))
470     SOUND S,2
480     IF DI<>O GOTO 420
490     FUN%=22
500     SOUND S,1
510     CALL PCL812 (FUN%,DAT%(0),ER%)
520     DI = DATA%(O) + DAT%(1)*256
530     DAT%(0)=0
540     DAT%(1)=15
550     COUNT#=DI
560     I=0
570     FUN%=1
580     CALL PCL812 (FUN%,DAT%(0),ER%)
590     FUN%=3
600     CALL PCL812 (FUN%,DAT%(0),ER%)
610     AD(I+1)=DAT%(0)*CC(I+1)
620     IF I<4 GOTO 650
630     IF DAT%(0)<300 GOTO 650
640     COUNT#=COUNT#+2!^(I+12)
650     I=I+1
660     IF I=16 GOTO 680
670     GOTO 590
680     IF COUNT#=0 GOTO 490
690     IF COUNT#=COUNTP# GOTO 720
700     COUNTP#=COUNT#
710     GOTO 490
720     SOUND 293,10
```

functions are locally monotonic. Since the step size may be refined without limit, the accuracy of the functions is limited only by the stability of the VCO and the resolution of the instrument. It is easy to approximate (through programmatic computation) the inverse of these functions. From the example shown in Figure 4, the control parameter (column 1) would become a function of the applied voltage or of the frequency counted by the system. A simple linear interpolation on data accomplishes this, with accuracy improving as the step delta decreases. It is this capability of precise functional inversion that is the basis of the operation of the function analyzer in its most powerful mode: the indirect mode.

Compare the direct data shown in Figure 4 with that listed in Figure 5, which illustrates data acquired using the indirect mode. In Figure 5, the VCO steps from 60 MHz to 70 MHz in 1 -MHz intervals. For each step, the instrument estimates, using the direct data of Figure 4, the control parameter

required to hit the target frequency [column 9). Once it is computed, the control parameter is applied and the resulting frequency is measured (column 2). Column 6 of Figure 5 is devoted to a logarithmic measure of the error between the desired frequency of each step and the actual measured frequency. This column indicates an average error of less than 1 part in 10^4, or a few kilohertz, for control of the 60-MHz oscillator. This accuracy will improve if the data in Figure 4 has a finer resolution between adjacent steps. Figure 5 documents the operation of the function analyzer as a frequency synthesizer in which a VCO is brought under precise numerical control.

OPERATION

It is important to note that calibration data like that shown in Figure 4 can be generated in seconds for any stable VCO. Once this data is measured and captured, the inexpensive VCO has become a versatile frequency synthesizer and can be considered a "calibrated" test instrument. The capabilities of this elementary piece of hardware can then be maximized by using menu-driven software.

Figure 6 shows a menu that has a variety of useful keyboard operations for control and measurement of a VCO. The first seven items enable the operator to set the period of the counter gate, set the output level of any of the D/A converters, select a frequency channel, read the analog channels, read a count or frequency, and calibrate the ADC channels. Items 8 and 9 are the basic calibration sweeps for VCOs under control of DA0 and DA1, respectively. Item 8 provides linear stepping of DA0 while reading the counter with double precision. It will also read and record the analog channels. Item 9 dispenses with double precision, but allows 100 samples to be acquired for calibration under control of DA1.

After calibration of a test oscillator using items 8 or 9, indirect operation follows using item 10 or 11, respectively. With item 11, the frequency may be stepped using a linear delta up to 250 times. Frequency and four separate voltages are recorded during each of these steps. Items 13 and 14 are identical to item 11, except that no frequency measurement is made during each step. This drastically accelerates the speed of a sweep. A linear sweep is used for item 14, and a logarithmic sweep for item 13. These two routines can collect up to 250 points in less than ten seconds. Figure 7 shows the operation of item 13, in which an ICL8038 audio oscillator (Figure 3b) is swept chromatically over the octave from 5 kHz to 10 kHz in 12 half tones. Note that the frequencies displayed are the target since they aren't measured. This frequency can be verified, however, by operation of item 11 over the same band. This would show that an accuracy of 1 part in 10^3 can be obtained for audio

```
STEP OUTPUT VARIABLE WITH DA1
FREQ CHANNEL=8
OUTPUT VARIABLE NUMBER 4
GATE=1
LIMITS 5.637421 E+07        7.313768E+07
ENTER FIRST VALUE? 60000000
ENTER LAST VALUE? 70000000
ENTER # STEPS? 10

 DA       FREQ.        AD     AD     DY/DX   ACC   STAB  MAG   TEST
2386.1  59982400.0    2.89   2.89    0.00   -36     0     0   60.0000
2480.7  60956230.0    3.01   3.01    8.38   -32   -18     0   61.0000
2553.8  61993280.0    3.09   3.09   12.59   -40   -18     0   62.0000
2624.6  63021890.0    3.18   3.18   11.18   -35   -18     0   63.0000
2700.7  63998080.0    3.27   3.27   10.61   -46   -19     0   64.0000
2788.4  64996930.0    3.38   3.38    9.37   -44   -19     0   65.0000
2886.5  66004030.0    3.50   3.50    8.66   -43   -19     0   66.0000
2995.6  67000510.0    3.64   3.64    7.09   -52   -19     0   67.0000
3118.1  67998020.0    3.77   3.77    7.35   -46   -19     0   68.0000
3254.3  68997760.0    3.94   3.94    6.07   -45   -19     0   69.0000
3405.1  69994880.0    4.12   4.12    5.42   -42   -19     0   70.0000

STATISTICS
MEAN
2835.8  644994910.0   3.44   3.44    7.88   -42   -17     0   65.0000

SIGMA
313.3     3168230.0   0.38   0.38    3.25     6     5     0   3.1623
```

Figure 5—*Indirect* control of *the BA1404 via external varactor tuning. The target frequency is stepped from 60 MHz to 70 MHz in 10 steps.*

```
STEP OUTPUT VARIABLE AGAIN? (Y/N) N
        (1) SET PACER
        (2) SET DA0,DA1
        (3) SET FREQUENCY CHANNEL (D/O)
        (4) READ D/I
        (5) READ A/D
        (6) READ COUNT
        (7) CALIBRATE A/D
        (8) STEP DA0 (DOUBLE PRECISION)
        (9) STEP DA1
       (10) STEP OUTPUT VARIABLE WITH DA0 (DOUBLE PRECISION)
       (11) STEP OUTPUT VARIABLE WITH DA1
       (12) SELECT OUTPUT VARIABLES
       (13) SWEEP OUTPUT VARIABLES WITH DA1 (NO FREQ MEASUREMENT)
       (14) STEP OUTPUT VARIABLES WITH DA1 (NO FREQ MEASUREMENT)
       (15) DISPLAY GRAPH
       (16) SET PARAMETERS
       (17) WRITE FILE DATA0
       (18) WRITE FILE DATA1
       (20) EXIT
```

Figure 6—*Menu for keyboard control-direct and indirect-of a VCO.*

```
STEP OUTPUT VARIABLE WITH DA1
Q=3
LIMITS          15938.17                4557.077
ENTER FIRST VALUE? 5000
ENTER LAST VALUE? 10000
ENTER # STEPS? 12
X1=1118         X2=1118         D/O=2

DA          FREQ.       AD          AD          DY/DX       ACC         STAB        MAG         TEST
3882.1      5000.0      7.57        7.57        -0.00       -1          -1          0           1.0000
3802.9      5297.3      7.42        7.42        0.00        -1          -17         0           2.0000
3719.0      5612.3      7.26        7.26        0.00        -2          -18         0           3.0000
3630.1      5946.0      7.09        7.09        0.00        -2          -18         0           4.0000
3537.0      6299.6      6.90        6.90        0.00        -3          -18         0           5.0000
343808      6674.2      6.71        6.71        0.00        -4          -18         0           6.0000
3334.0      7071.1      6.50        6.50        0.00        -4          -18         0           7.0000
3222.3      7491.5      6.29        6.29        0.00        -5          -18         0           8.0000
3106.6      7937.0      6.07        6.07        0.00        -6          -18         0           9.0000
2984.1      8409.0      5.82        5.82        0.00        -8          -19         0           10.0000
2851.0      8909.0      5.57        5.57        0.00        -10         -18         0           11.0000
2710.4      9438.7      5.28        5.28        0.00        -13         -18         0           12.0000
2560.8      10000.0     5.00        5.00        0.00        -33         -19         0           13.0000

STATISTICS
MEAN
3290.7      7237.4      6.42        6.42        -0.00       -7          -17         0           7.0000

SIGMA
410.5       1556.7      0.80        0.80        0.00        8           5           0           3.7417
```

Figure 7— *Indirect control of ICL8038* **(chromatic scale).**

frequencies synthesized in this manner.

The PC brings to bear substantial processing and other capabilities which enhance the utility of the function analyzer beyond mere acquisition and display of calibrated measurements. For example, column 6 in Figure 5 gives a logarithmic measure of the accuracy with which one variable approximates another. The variables to be compared (usually the target and measured frequencies] are selected by menu. Column 7 in Figures 4 and 5 compares consecutive values of any selected variable; if this variable is supposed to be constant, the column measures its stability in parts per unit.

As shown in Figures 5 and 7, after every sweep in indirect mode each column is processed for mean and standard deviation. Suppose, for example, that after an initial calibration routine, a VCO is measured via item **11** at a fixed frequency 100 consecutive times. Then the sigma value of column 2 is an excellent measure of the frequency stability of this VCO. This sophisticated and

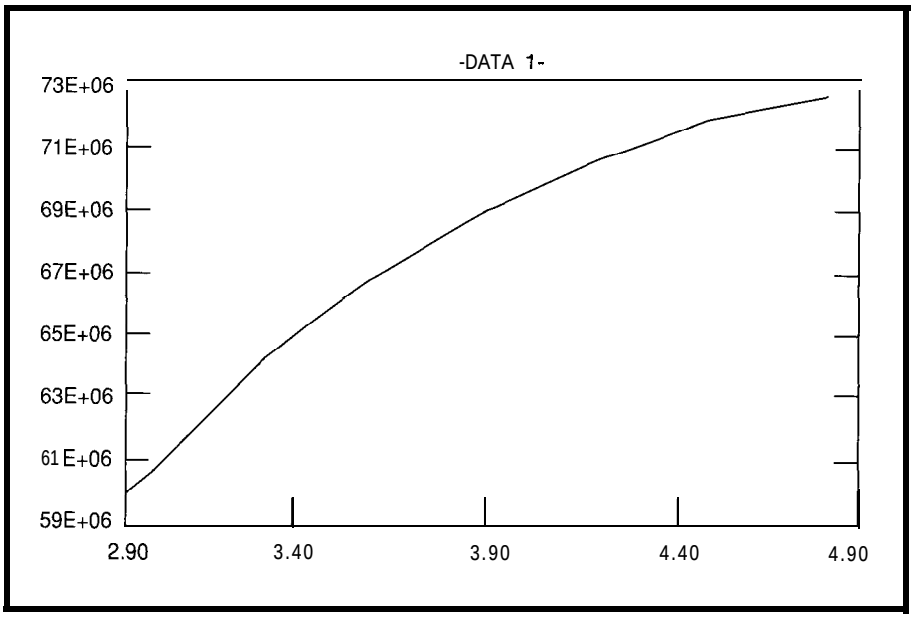


Figure 8—*Indirect* control *of BA 1404. Here, the varactor voltage, as* measured *by the* calibrated *ADC channel, is stepped in 50-m V* increments.

formerly tedious measurement (known as the *Allan variance)* is performed automatically.

In all examples so far, we have illustrated the control of frequency in direct and indirect mode. It is easy to see, however, that any variable may play the role of frequency. This flexibility means that in a complex circuit, virtually any two variables which vary with one another can be plotted, one as a function of the other.

This idea is illustrated in Figure 8, which shows the operation of a 60-MHz oscillator. In this case, the varactor voltage is a function of the control parameter. The control voltage is produced by the function analyzer and is stepped in 50-mV increments. This experiment shows the frequency of the oscillator (displayed in this figure) as a function of the stepped varactor voltage.

Figure 9 shows a menu from one of the programs I wrote as part of the function analyzer. This program facilitates the construction of a wide variety of interesting variables after data acquisition. For instance, the current through a resistor may be computed using selection 19. In this case, A and B are a pair of calibrated voltage readings that were taken at the terminals of the load, and C is the resistance of the load. For another example, selection 16 computes a derivative as a quotient of differentials, where the differentials may be those of any two selected variables. These are but two examples of the postprocessing capabilities provided by this utility. Careful examination of the menu in Figure 9 can give you an idea of the program's other features. Limited only by the constraints of monotonicity and stability, the function analyzer can plot any of these variables as a function of any other.

CONCLUSION

During this article, I have emphasized the measurement and control of frequency. I did this to show you examples of the primary use of the function analyzer. Now that you see the scope and sophistication of the measurements that can be made with the function analyzer, I hope you have as much fun applying it as I did designing it. PCs can produce musical notes. It is easy to program the instrument so that it "sings along" with itself as it hits various frequencies in indirect mode. The full utilization of the PC will change the nature of electronic test equipment. ❑

David Birkett is currently writing a book on mathematics, personal computers and electronics.

```
STEP WHICH OUTPUT VARIABLE?
        (1) DA
        (2) COUNT
        (3) FREQ
        (4) FREQ*64
        (6) AD(t)
        (7) AD(2)
        (8) AD(3)
        (9) AD(4)
        (10) TARGET
        (11) INDEX
        (12) CONSTANT
        (13) 0
        (16) DY/DX
        (17) ACCURACY
        (18) STABILITY
        (19) (A-B)/C
        (20) MAGNITUDE

ENTER SELECTION?4
ENTER DY VALUE?4
ENTER DX VALUE?6
ENTER FIRST ACCURACY VARIABLE4
ENTER SECOND ACCURACY VARIABLE10
ENTER STABILITY VARIABLE4
NTER A VARIABLE
NTER B VARIABLE
NTER C VARIABLE
NTER CONSTANT FN1
RINT WHICH FREQ4
RINT WHICH AD FIRST6
RINT WHICH AD SECOND7
RINT WHICH TEST VARIABLE1 1
'AKE WHICH MAGNITUDE

SCALE VARIABLES
VHICH VARIABLE?
AULTIPLY BY
VHICH VARIABLE?
AULTIPLY BY
```

Figure 9-Menu for construction *of useful variables from primary acquired data. After construction, any variable may be listed as a function of any other variable subjected to indirect control.*

SOFTWARE

Software for this article is available from the Circuit Cellar BBS and on Software On Disk for this issue. Please see the end of "ConnecTime" in this issue for downloading and ordering information.

CONTACT

B&C Microsystems, Inc.
750 North Pastoria Ave.
Sunnyvale, CA 94086
(408) 730-5511
Fax: (408) 730-5521

I R S

404 Very Useful
405 Moderately Useful
406 Not Useful

Neural Network Basics

FEATURE ARTICLE

Dwayne Phillips

You don't have to be a surgeon to understand how neural networks function. Dwayne introduces the Adaline and Madaline, two fundamental concepts in neural network design.

The field of neural networks supports an approach to solving difficult problems using many small, simple processors working in parallel. Neural networks may seem like they are the stuff of science fiction, but they are reality; even better, they are simple to implement.

Neural networks are more popular now than in the recent past, but they are certainly not new. They were first discussed around 30 years ago (see the reference list at the end of this article), and they were used to read characters and predict the weather long before we had megabytes of RAM and supercomputers.

I'll take you back to the beginning with a brief review of the basics of neural networks. Then I'll look at what they can do and teach you about their basic building blocks and how to implement them using simple C code,

FUNDAMENTAL CONCEPTS: RECOGNIZING PATTERNS AND THE HUMAN BRAIN

Neural networks are especially adept at solving problems of pattern recognition. There are many problems that are trivial for people to solve but confound traditional computer approaches. Examples of these kinds of problems are interpreting images ("see Dick, see Jane, see Dick run"), predicting weather ("red sky at morning, bring a rain coat"), and reading handwritten characters (except those appalling scrawls by doctors). Conventional approaches to these problems require supercomputers and teams of expert programmers.

Since people could solve these pattern recognition problems easily, researchers studied the structure of the human brain. In the human brain there are many processing elements called *neurons.* Each neuron is connected to many of its neighbors through connections called *synapses.*

The researchers used this model to develop neural networks. Figure 1 shows a diagram of a generic neural network. The neural network takes inputs on the left and produces outputs on the right. In the middle are several layers of processing elements connected together with a layered-mesh network. Each processing element is small, simple, and not very powerful by itself. The power comes from the combined effect of many processing elements because of the way that they all work together in parallel on the "problem."

Neural networks "learn" to recognize patterns. This capability is what makes them unique and also contributes to their mystique. In order for a neural network to learn, you must supply the neural network with a set of training data. This training data should be typical of the problem under consideration and should contain

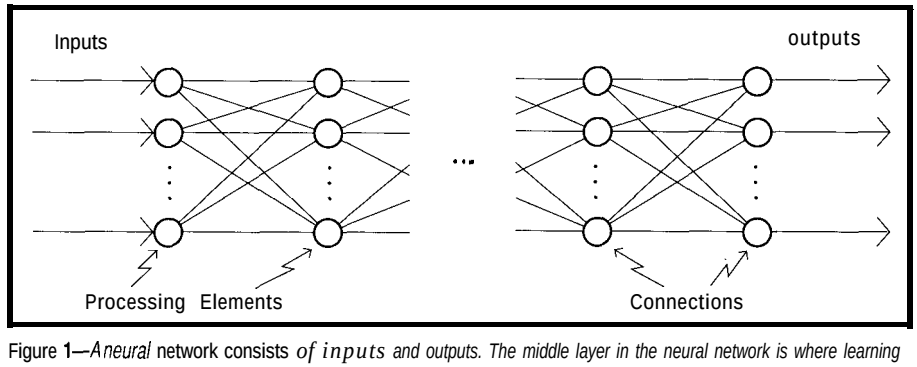


Figure 1—*A neural* network consists *of inputs and outputs. The middle layer in the neural network is where learning to* recognize patterns occurs,

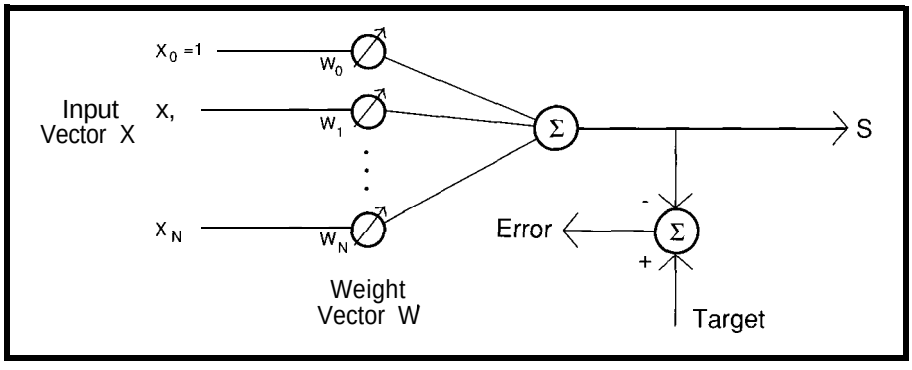


Figure 2—*The* Adaptive *Linear combiner is analogous to the* ***AND*** statement in *digital* computers. It is *the* basic building *block of all neural networks.*

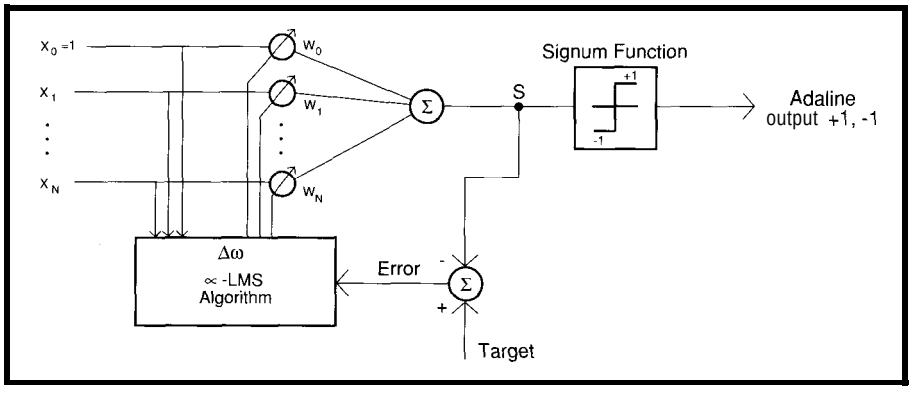


Figure 3—*An* elementary neural network known as *Adaline* distinguishes patterns that are linearly separable.

results that are known to be correct answers. You begin by feeding one set of training data into the neural network and then examining the output. If the output is wrong, you adjust the synaptic weights of the connections in the neural network. You repeat this process of feeding data into the network, examining the output, and adjusting the weights until the neural network's output matches the known correct answer. This is called *supervised learning.* When the learning (or training) is complete, you store the adjusted weights. These weights contain the "knowledge" that will allow the neural network to recognize patterns that were not in the training data. The skill in manipulating neural networks is related to the level of skill you use when you adjust the synaptic weights.

During the 1960s, neural networks were built using discrete hardware components. The processing elements consisted of simple voltage adders, and the connections between the elements were made through a type of variable resistor. Researchers fed problems to these systems, and feedback loops were employed to adjust the resistors until the neural network matched the correct answers. With time, all things progress. Now, you can buy neural network chips that are electrically alterable. What makes these devices particularly useful is that after you train them, they will store the weights that produce correct answers.

Another method in wide use today is to simulate neural networks in software on traditional computers. The simulation is a program that implements neural network equations like those I'll show you a little later. As the code segments given below will demonstrate, these programs are simple and typically involve array multiplication and addition.

These neural network simulation programs have one great advantage over traditional computer programs—they are extremely adaptable. Since neural networks can learn (and can be retrained), you write a neural network program once, then retrain it to teach it how to solve new problems. Traditional computer programs are algorithmic-meaning you code the algorithm in a programming language. The algorithm, however, is different for each problem, so you must write a new program for each new problem. Neural network programs are adaptive-you write the neural network program once, and adapt it by storing different sets of weights that solve different problems.

THE THREE STEPS IN USING A NEURAL NETWORK

There are three steps in using a neural network input data, train the network, and use the network on new problems. The first step could take the longest depending on the method you use to input the data. Neural networks can only work with numerical data. If you have any textual data you want to use, you must first transform it to numbers. Your neural network program may query a user for input data, or you may write routines to read data from database or spreadsheet files.

Training is the process of running the training data through the neural network and adjusting the network until its output matches known correct answers. The training data must have known correct answers. The neural network adapts itself, or learns, during the training phase. The algorithm the neural network uses to adapt itself is called the *learning law.*

To process new cases, you input new data into the neural network and wait for the answer. If the data for the new cases are similar to the training data, then the neural network will produce correct answers. The key to accurate performance is the training data. You need many representative sets of data for training. How many samples you will need, and how representative they must be, is open for experiment.

THE ADAPTIVE LINEAR COMBINER

Figure 2 shows an *adaptive linear combiner.* The adaptive linear combiner is the basic building block of all neural networks. It is to neural networks what the AND gate is to digital computers. The adaptive linear combiner multiplies each element of an input vector by corresponding elements of a weight vector and

finally calculates the sum of these products.

Listing 1 shows a small snippette of C code that implements the adaptive linear combiner (the complete source and executable code for the Adaline and Madaline programs are available on the Circuit Cellar BBS). This is a simple array multiplication and addition. Equation 1 describes the adaptive linear combiner and is defined as:

$$s = \sum_{i=1,N} x_i \times w_i + w_0 \qquad (1)$$

$$x_0 = 1 \text{ (always)}$$

The arrow facing toward the left side of Figure 2 shows an error calculation. The error is the difference between the output and the target, or correct answer. We will use the error to adjust the values of the weight vector w. The adjusting process is how the neural network learns.

THE ADAPTIVE LINEAR ELEMENT —ADALINE

Figure 3 shows an example of one type of elementary neural network—an ***adaptive linear*** element-which is also known as an *Adaline*. The Adaline is a linear classifier, which means it can distinguish patterns that are linearly separable. Figure 4 shows an example of two classes that are linearly separable. Figure 5 shows the numbers that were used to create the graph in Figure 4.

Suppose we had data on the success of students at a summer computer camp. The result could be that shown in Figure 4. The students who scored high on both math and ccience tests fared well in our computer camp.

The two classes of students in this example are linearly separable since I can draw a straight line that separates the classes. The Adaline can adapt to this data and then predict the success of future computer camp students based on their math and science scores.

The Adaline has the adaptive linear combiner plus these two new parts: the *signum* ***function*** and the ***alpha-LMS learning algorithm.*** The

Listing 1—*A short routine in C* ***illustrates the implementation of the*** Adaptive Linear Combiner.

```
long adaptive_linear_combiner (long *s, long N,
                               long w[], long x[])

        long            i;
        *s = 0;
        for (i=0; i<N+1; i++)
                *s = *s + w[i]*x[i];
```

signum function takes the sum-of-products s and transforms it into a binary output of +1 or -1. The output of the signum function is +1 if the input is greater than or equal to zero, and -1 otherwise. The code in Listing 2 shows the subroutine that implements the signum function. The Adaline can only have a +1 or -1 output because the Adaline separates or classifies data into two classes. You must design your problem to fit this limitation.

The alpha-LMS learning algorithm is how the Adaline learns or adapts its weights to the correct answer. It considers the error and the x-inputs and automatically produces the necessary changes in the weights. The alpha-LMS algorithm works by trying to minimize the mean square error in the training patterns, hence the name LMS [least mean square).

$$\Delta w_i = a \times x_i \times \text{error} \quad \text{for } i = 1,N \qquad (2)$$

error = target-s

0.1 <a< 1.0

$$w_i = w_i + \Delta w \quad \text{for } i = 1,N \qquad (3)$$

Equations 2 and 3 describe the alpha-LMS algorithm. They illustrate how to multiply the error by an input and an ***alpha factor*** to produce the change in each weight. The factor ***alpha*** is a constant that is set between 0.1 and 1.0. If you set ***alpha*** too high, the learning process will generate oscillations and never reach an end. If you set it low, the learning will progress very slowly. You must experiment with this. The code in

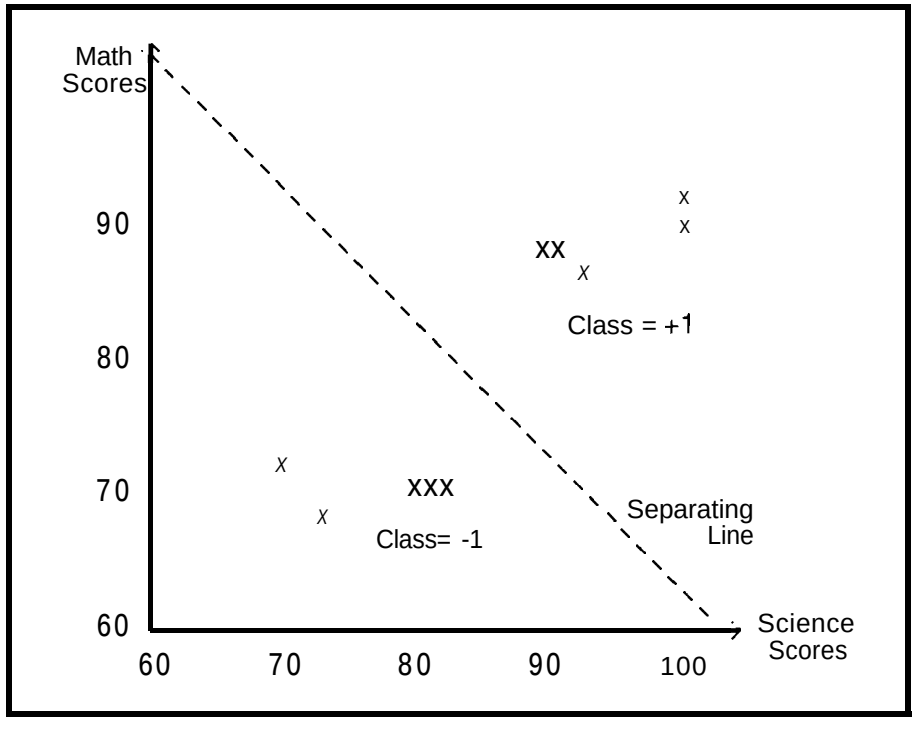


Figure 4—*Adaline* iscapable *of* distinguishing students who scoredhigh in math andscience, *depending on* which *side of the line they fall.*

Listing 2—*The signum function uses a binary output of + 1 or -1 to establish the classification of data being used, a characteristic of Adaline.*

```
long signum_function(long input)

        long result = 1;
        if (input < 0) result = -1:
        return(result);
```

Listing *3-An example of how Adaline adapts ifs weights to the correct answer using alpha-LMS is shown.*

```
long alpha_LMS (long target, long s, float alpha,
                        long w[], long x[], long N)
{
        long delta-w, i;
        for  (i=0; i<N+1; i++){
                delta-w = alpha*x[i]*(target-s);
                w[i]    = w[i] + delta-w;
```

Listing 3 shows a simple routine written in C that implements the alpha-LMS algorithm.

Equations 2 and 3 and the code in Listing 3 are the "magic potion" that allows neural networks to learn. They form the basic feedback loop that adjusts the weights until the network produces correct answers. You can initialize the weights to small random numbers, then use the alpha-LMS algorithm as a feedback loop to dynamically adapt them.

The alpha-LMS algorithm uses the principle of minimal disturbance, which states "...Adapt to reduce the output error for the current training pattern, with minimal disturbance to responses already learned..." [Widrow].

Math Score	Science Score	Class (+1 or −1)
92	100	+1
90	100	+1
88	90	+1
88	92	+1
86	94	+1
72	70	-1
68	72	-1
70	80	-1
70	82	-1
70	84	-1

Figure *5-Taking the data from Figure 4, tabulating if, and feeding if fo the Adaline allows the neural network to "learn" the desired task.*

In other words, adapt the weights, but do not disturb them too much.

AN ADALINE EXAMPLE

I'll use the data in Figure 4 and Figure 5 as an example of how to use an Adaline. First run the full Adaline program and interact with menus until you are comfortable with the interface. The first step when using this program is to enter the data. For this exercise, type in the 10 sets of data given in Figure 5. The file names you choose are completely arbitrary. The next step is training the network. Choose this option and watch as the Adaline loops through the data until it can produce correct answers for all 10 inputs.

The final step is processing new data cases not included in the original training set of Figure 5. Enter any set of points and see how the Adaline classifies it. You can easily find a case where the Adaline gives a wrong answer. This is because we only used 10 sets of data for training. The more data that is used in training the network, the better the Adaline will perform.

THE MULTIPLE ADAPTIVE LINEAR ELEMENTS-MADALINE

Figure 6 shows a more complex and capable neural network using

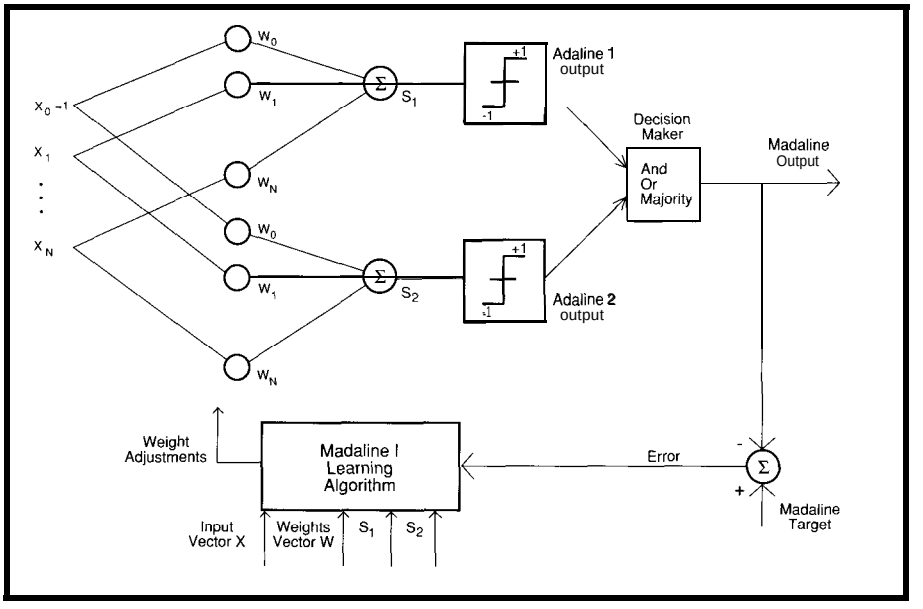


Figure 6—*The use of multiple Adalines is known as Madaline.*

multiple Adalines-this kind of network is known as a Madaline. The Madaline in Figure 6 contains only two Adalines, but it could have as many as you want. I built the Madaline using the adaptive linear combiner and the Adaline. Therefore, I can use code segments 1, 2, and 3 from those devices. The Madaline is more capable than the Adaline because it can separate patterns that are not linearly separable.

Figure 7 gives an example of patterns that are not linearly separable. Consider the data in Figure 4 and Figure 5, and suppose the computer campers at the high and low ends of the math and science scores did better than the rest in computer music. The bent line drawn in Figure 7 shows the two classes of computer music campers. If we trained the Madaline to separate these patterns, we could use future math and science scores to predict computer music success.

The Adaline cannot distinguish the two classes shown in Figure 7. Recall that the Adaline can only distinguish classes that are separable by a single, straight, dividing line.

The Madaline has two new components that the Adaline did not have: the ***AND, OR, MAJORITY decision maker*** and the ***Madaline-Z*** learning algorithm. The AND, OR, MAJORITY decision maker decides the final output. It takes the outputs of each Adaline (+1 or **-1**) and determines the final output of the Madaline (+1 or **-1**). If you use the AND decision maker, it logically ANDs the +1 and **-1** inputs. If you use the OR decision maker, it logically ORs the +1 and **-1** inputs. The MAJORITY choice counts the number of +1 and **-1** inputs and chooses the most popular input. The AND, OR, MAJORITY decision maker is a simple device you can build using Adalines. The code in Listing 4 implements the decision maker in software.

The second new part of the Madaline is the Madaline-I learning algorithm shown in Listing 5. In the Adaline, we used the alpha-LMS algorithm to adjust the weights of an Adaline. The Madaline is a little more complex, however. How are the weights of this network adjusted? Are the weights of only one, several, or all of the Adalines adjusted? Which Adaline is adjusted? How is that done?

The Madaline-I algorithm (you may be interested to know that there are three different Madaline algorithms) adjusts the weights of the Adaline whose +1 and **-1** output disagrees with the Madaline target, and whose s output is closest to zero. For example, suppose the Madaline had five Adalines and the Madaline produced an incorrect output of +1 (the correct target was **-1**). The Madaline-I algorithm looks at the Adalines that had outputs equal to +1. **Next, it** examines their s outputs (equation **1)** and finds the Adaline whose s is closest to zero. Now it adjusts the weights of that Adaline using the alpha-LMS algorithm shown earlier. Train the Madaline by using your input data sets and adjusting the Adaline weights until it produces correct answers for all cases. Implementing the Madaline-I is a simple matter of coding the algorithm in Listing 5.

A MADALINE EXAMPLE

I'll use the data in Figure 7 and Figure 8 as an example of how to use a Madaline. First, run the Madaline program and interact with it through the menus. Next, enter the data given in Figure 8. The file names you choose are arbitrary.

The next step is training. You must specify how many Adalines you want in the Madaline. I chose seven, but feel free to experiment. The more Adalines in the network, the more power it has, but this also slows down the performance of the network. You must also select the AND, OR, or MAJORITY decision maker. I chose MAJORITY. After choosing your options, watch as the Madaline loops through the data until it can produce correct answers for each of the **10 sets** of data.

The final step is processing new data cases not in the original training set. Enter any set of points and see how the Madaline classifies it. Once again, it is easy to select points that produce a wrong answer because we only used **10** sets of data for training.

A REAL-WORLD EXAMPLE: LOAN APPLICATIONS

The previous examples with computer campers gave you an idea of the use of neural networks, but now let's move to something more concrete-money. An excellent use of neural networks is examining loan applications and deciding which loans to approve or disapprove. Loan officers often make poor decisions because, being human, they sometimes consider nonessential factors that can mislead them. Considerations such as

appearance, dress, speech, politics, and family connections can sway loan officers, and sometimes the result is a bad loan.

The loan application problem is well suited for neural networks for several reasons. First, there is an abundance of training data. Banks keep their completed loan application forms, and of course they remember whether or not an approved application was successful.

This history will be the training data set. Also, the loan application data are either numerical or can be easily transformed into numbers. Facts such as age, income, and debt are numbers, and you can transform occupation, address, and credit record into number-codes very easily,

Figure 9 gives sample data sets for the loan application example. The bank has this information for short-term $20,000 loans. Borrowers paid off five of the loans on time [these are the +1 cases), and defaulted on five of the loans [these are the -1 cases). I'll use these as training data and then predict the success or failure of future loans. If the network predicts a loan will be successful, then I'll approve that application.

The first step is to input the data in Figure 9. Then run the Madaline program [this is a nonlinear problem so the Adaline cannot solve it) and enter the 10 sets of data. Each set of data has four elements and a target. We plotted the data sets and separating lines for the two previous examples on X-Y axes because they only had two elements per data set. Plotting the data of Figure 9 requires a drawing with four axes and a separating object. I'll

Math Score	Science Score	Class (+1 or −1)
72	70	+1
68	72	+1
92	100	+1
90	100	+1
88	90	-1
88	92	-1
86	94	−1
70	80	-1
70	82	-1
70	84	-1

Figure 8—*Data can be used in conjunction with Figure 7 to run the Madaline program.*

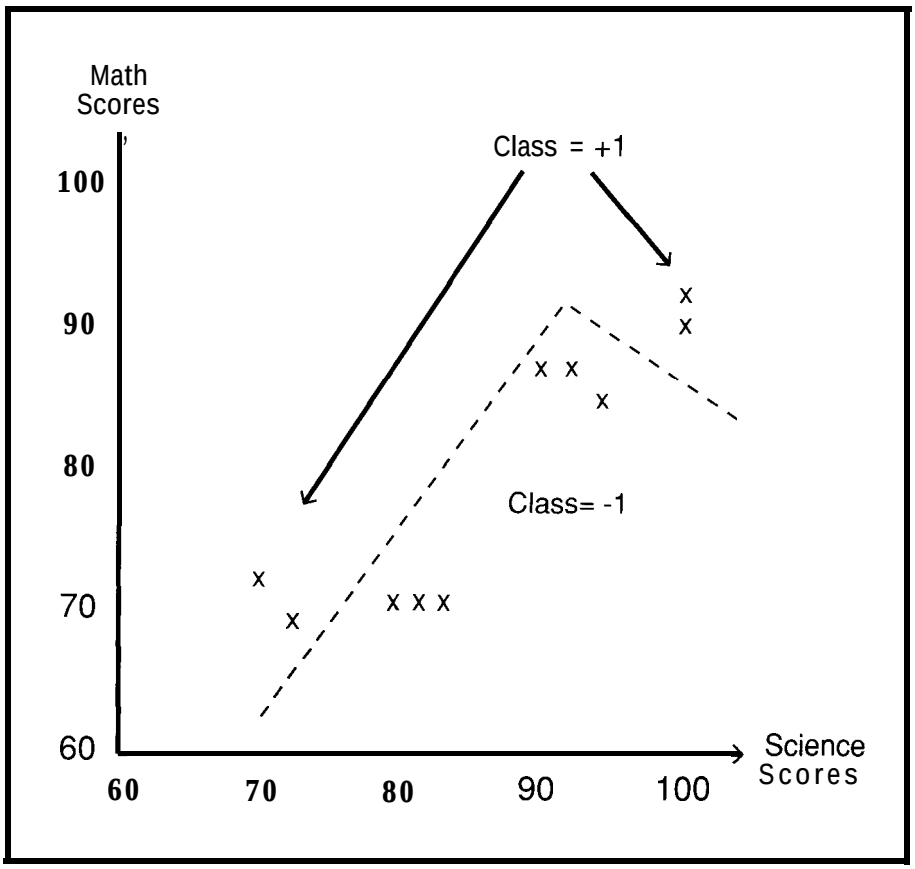


Figure 7-When using Madaline, *the* capability of differentiating *patterns* not linearly separable is possible.

Listing 4—*Learning to* use *the AND,* OR, *MAJORITY decision maker is straightforward.*

```
long  and_or_majority(long outputs[], char choice, long A)

        int     i, minus = 0, plus = 0;
        long    result = -1;


        /* AND */
        if (choice == 'a' || choice == 'A') {
                result = 1;
                for (i=0; i<A; i++)
                        if (outputs[i] == -1)
                        result = -1;
        }

        /* OR */
        if (choice == 'o' || choice == 'O') {
        for (i=0; i<A; i++)
                if (outputs[i] == 1)
                        result = 1:
        }

        /* MAJORITY */
        if (choice == 'm' || choice == 'M') {
                for (i=0; i<A; i++) {
                        if (outputs[i] == 1) plus++;
                        if (outputs[i] == -1) minus++:

        if (plus > minus) result = I:
        }
        return(result);
```

leave that as an exercise for the reader-it is possible.

The next step is training. I used seven Adalines in the Madaline and the MAJORITY decision maker. The Madaline learned the training data in less than a minute.

The final step is processing new loan applications. Once again, it is easy to enter an example that produces a "wrong" answer. I quoted wrong in this case, because with this 4-dimensional problem I am not sure which domains are right or wrong answers. My advice is train the network with a minimum of 20 or 30 sets of data, especially if you are a loan officer.

Income $k	Age	#Years Employed	Current Debt$k	Approve? yes=+1 no=-1
60	40	15	5	+1
55	35	13	7	tl
50	30	8	3	+1
70	42	20	12	+1
65	45	22	10	+1
30	40	15	20	-1
25	22	1	1	-1
20	30	7	10	-1
22	25	2	15	-1
15	20	1	1	–1

Figure 9-A *real world application consists of data important to a loan officer.*

CONCLUSION

I've discussed the basics of neural networks and showed how to use them. There are three properties to remember about neural networks. They can be used to recognize and separate patterns, are simple to program, and are adaptable.

Neural networks (and people] perform pattern recognition tasks much better than traditional computer approaches. Simulating a neural network in software is not difficult. The short, simple code listings given here show the heart of the neural network. Once you have the neural network program, you use it to solve many different problems. You train the network to solve new problems just like I did when I trained the Madaline to solve the problems I illustrated.

Most of all...definitely do experiment! Use the Adaline and Madaline

programs on new problems. These programs use integer arithmetic and are very flexible. Predicting stock market prices, predicting the weather, and optical character recognition are good experiments. ❑

Listing 5—*In the Madaline I* Learning *Algorithm, weights are adjusted in the Adalines that disagree with the Madaline target.*

```
Do until Madaline's output = target for all training cases

      Do for i=1,number of training cases

            Do for j=1,number of Adalines in Madaline

                  calculate s[i] for Adaline[j] (code listing 1)
                  calculate output[i] for Adaline[j] (listing 2)


            calculate Madaline output (code listing 4)
            If Madaline output != target Then
            {
                  Find the Adaline whose output != target
                  and whose s is closest to zero

                  Use the alpha-LMS learning algorithm
                  on that Adaline (code listing 3)

                  Repeat all the training for all the
                  training cases
```

Dwayne Phillips works as a computer engineer with the U.S. Department of Defense. He has a Ph.D. in Electrical and Computer Engineering from Louisiana State University.

SOFTWARE

Software for this article is available from the Circuit Cellar BBS and on Software On Disk for this issue. Please see the end of "ConnecTime" for downloading and ordering information.

REFERENCE

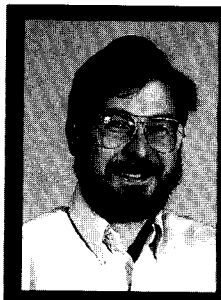
"Thirty Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropogation," Bernard Widrow, Michael A. Lehr, Proceedings of the IEEE, Vol. 8, No. 9, September 1990, pp. 14151442.

I R S

407 Very Useful
408 Moderately Useful
409 Not Useful

DEPARTMENTS

Ticks, Pops, and Restarts: The '386SX Project Gets A BIOS Extension



The PC's built-in BIOS doesn't always do all the tricks you need. When you have some custom hardware, it's time to add some custom software. Find out how to add a BIOS extension and check out another Nisley war story.

FIRMWARE FURNACE

Ed Nisley

Traditional embedded systems are ready to go at the flip of a switch. The PC's built-in ROM BIOS lights up immediately, but, as you saw in the last issue, loading code from diskette can take quite a while. There's no cheap cure for that, but making your code an extension of the BIOS can help.

In this column I'll explore BIOS extensions in more detail with a set of routines that capture interrupts, support the Firmware Development Board's power failure signal, and record information in nonvolatile storage. You can use the EEPROM or battery-backed RAM circuits from the last two issues as long as the firmware can enable and disable the –WE line.

As a special treat for those of you who think this embedded systems stuff is all too easy, I'll also tell the tale of a fascinating bug in INKnet's serial interrupt handler. It waited three years to pounce on me...and you may have it in your ISR code, too!

THE KEY TO THE CODE

Figure 1 shows the new hardware that's needed for this column: a push-button switch with a pull-up resistor on bit 9 of the port at 31C. It's barely worth warming your soldering iron, but every now and then we need an easy one....

Holding the button can be awkward at times, so I also rewired the front-panel lock switch in parallel with the button. Because we haven't used the keyboard yet, I figured the lock was superfluous enough to usurp for a second purpose.

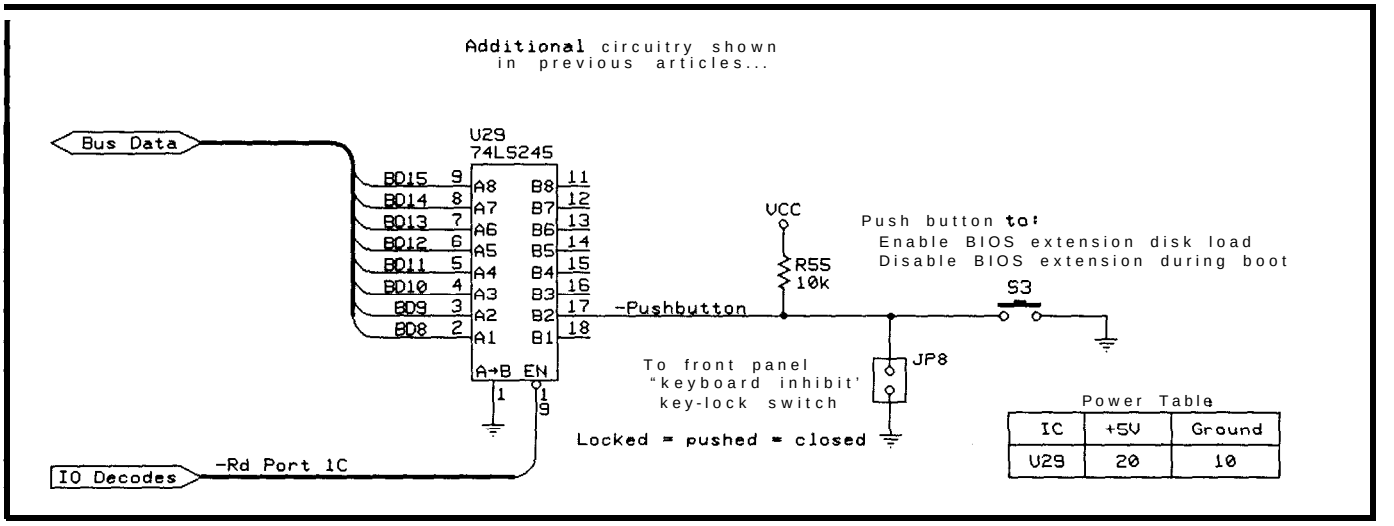


Figure 1—*This* **month's** *hardware, a button and a resistor, allows you to skip over a new BIOS extension at bootup should if continually crash the system.*

Should you ever come up against a "locked" clone, just whip out your Swiss Army knife's Phillips blade, unscrew the clone's case, yank out those pesky little wires on the lock switch, and you're on the air. I trust I'm not compromising the security of what was once the Free World by letting that trick out of the bag. The Original AT's lock switch disabled the keyboard and secured the metal cover to prevent just such an assault.

So much for the hardware. Now, on to the code..

EXTENSION ESSENTIALS

The Original PC BIOS didn't scan for extensions, which led to some truly remarkable kludges as each vendor devised unique and mutually incompatible ways to glue new functions into old PCs. The method used now dates back to a revision of PC BIOS slightly before the XT, so, for all intents and purposes, every PC handles it the same way now.

If you are designing video adapters there is a more recent technique to ensure your extension will be initialized first regardless of its address. The method, devised by IBM for its PS/2, is not supported by all BIOSs and is beyond the scope of my column, but check the references for more details. It might come in handy if you need control right away, but unless you're running a "true blue box," the compatibility issues warrant some care.

The part of FDBEXT. ASM shown in Listing **1 sets up** the 55 AA signature, length, and checksum bytes required for the BIOS scan. Recall that the checksum byte in the source code must be zero, because our diskette boot loader computes the value as it copies the extension into the Firmware Development Board.

The code at BootEntry is the escape hatch that I suggest you build into all your extensions, at least during debugging. If the switch is pressed [or the lock is on) while booting, **FDB EXT** updates the LEDs and returns to the BIOS. This can save your bacon if your new extension crashes the BIOS boot sequence. Trust me, it can happen.

Assuming the switch is off, the code shown in Listing 2 makes the

Listing 1— **This** *code is loaded into the battery-backed* **RAM** *at C800:0000* **on** *the Firmware* Development *board. The B/OS passes control to the instruction just after the length byte during the power-on sequence. The first thing this code does is check the pushbutton-switch bit;* if *the button is pressed the code returns to the BIOS.*

```
; This does not start at offset 0100, so be careful
; about data accesses!
; Actual execution is at absolute address C800:0004...

          CODESEG
          STARTUPCODE

          DB        055h              ; signature
          DB        0AAh
          DB        2                 ; length in units of 512 bytes
          JMP       SHORT BootEntry   ; force two-byte jump
          DB        00h               ; loader sets this value

; constants that must be stored within the checksummed region

RevCode   DW        0001h             ; current revision level

;- if pushbutton is down, exit without doing much

BootEntry:
          MOV       DX,STAT_ADDR
          IN        AX,DX
          TEST      AX,PUSHBUTTON
          JNZ       SHORT Continue    ; nonzero means not pushed

          MOV       AX,NOT 0101h      ; show - to track our path
          MOV       DX,LED_ADDR
          OUT       DX,AX
          RETF                        ; return to normal BIOS boot
Continue:
```

whole software development process I'm using for this series work correctly. As Steve puts it, "Let me explain.. "

THE CASE OF THE MISSING PSP

FDBE XT is written in Borland's Turbo Assembler using the TINY memory model to produce an ordinary COM file. As far as TASM and the linker are concerned, the program will run under DOS, so all the usual DOS assumptions and restrictions apply.

As you know by now, COM files are exact binary images of the program's code. They date back to the days of CP/M and 8080 CPUs when 64K of RAM was all you had even if you were a big spender. The operating system reserved the first 256 bytes of RAM to get control of the 8080's reset and interrupt vectors, so your programs were loaded at address 0 100.

MS-DOS adopted the same memory layout, except that 64K was suddenly not so much after all. A COM file fit neatly into one 64K segment atop the reserved 256 bytes, which, still filled with operating system stuff, became known as the Program Segment Prefix. Executable (EXE) files are handled differently: their PSPs live in a different segment, and we'll get to them later, but for now the key point is that COM files start at 0 100 for historical reasons.

Although all of the code and data addresses within a COM file assume that it's loaded at offset 0100, the actual disk file does not include those first 256 bytes. The instruction at program offset 0 10 0 is thus at 0 0 0 0 relative to the start of the file. DOS must set up the segment registers so the offsets are correct within the segment where the program is loaded.

The disk boot loader introduced in issue 3 1 simulates this process. It loads your program from diskette at address 10 0 0 : 0 10 0 with nothing in the first 256 bytes. Although there's no PSP, that trick let us use standard COM files without a specialized linker. As long as the program didn't expect anything in the PSP, its absence makes no difference.

BIOS, on the other hand, knows nothing of this. When it finds our Firmware Development Board extension, it passes control to that branch instruction with CS:IP set to C800: 0003. Because we'll be setting interrupt vectors as well as changing data, our code must somehow adjust all the segment registers.

The solution is a simple matter of subtraction. A given physical address can be accessed by many different segment and offset values. The CPU simply shifts the segment register left by four bits, adds the offset, and uses that as the physical address. At least that's the case in real mode, which is all we need for now.

The branch at C800: 0003 is at physical address C8003. It is also at C7F0:0103, because C7F00 + 00103 = C8003. Thus, if we reload the segment registers with C7 FO rather than C800, all our offsets are correct and we can use COM files for BIOS extensions.

The easiest way to reload both CS and IP is from the stack with a RET F (Far Return) instruction. Listing 2 shows the trick in all its glory.. .not very impressive to see, is it?

COM programs assume that CS, DS, ES, and SS all have the same value, but CS and DS are the key registers. I load DS and ES from the adjusted CS value, but SS cannot aim into the nonvolatile memory because it is normally write protected.

Fortunately, as long as we just PUSH, POP, CALL, and RET from the stack, whatever the BIOS uses for SS and SP will work fine. I haven't looked at how deep the default stack is, but you might want to check it out if you need lots of room for some reason.

Listing 2—*If the switch is open, the next step is to adjust the segment registers. The RETF instruction loads the new values info CS and IP from the stack. Note: This code uses the FS and GS segment registers found in '386 CPUs and will not run on earlier CPUs.*

```
;-adjust CS and DS to simulate the normal COM situation
; We need both code and data starting at offset 0100 rather
; than 0000C800:0000 is also C7F0:0100, so we just subtract
; 0010 from the segments. Storing this CS in the vectors
; allows normal access after an interrupt

        MOV     AX,NOT 0100h        ; show single here
        MOV     DX,LED_ADDR
        OUT     DX,AX

        MOV     AX,CS
        SUB     AX,0010h            ; adds 100 to offsets in segment
        PUSH    AX
        PUSH    OFFSET BootStart
        RETF                        ; set CS and IP to new values

BootStart:
        MOV     AX,0057Eh           ; show r0 on LEDs to mark entry
        CALL    ShowBits

        PUSH    DS                  ; preserve seg regs
        PUSH    ES
        PUSH    FS
        PUSH    GS

        MOV     AX,CS               ; set up DS to match CS
        MOV     DS,AX
        MOV     ES,AX

        MOV     AX,0                ; FS points to 0000:xxxx
        MOV     FS,AX

        MOV     AX,0040h            ; GS points to 0040:xxxx
        MOV     GS,AX

        CALL    OpenRAM             ; enable writes

        INC     [ES:ResetCtr]       ; count this reset
```

Listing 3—*The BIOS extension captures the BIOS timer interrupt to count the ticks since the most recent reset. The value of CS stored in the interrupt vector allows access to the extension's variables in nonvolatile memory.* Because this code was assembled in 386 *mode, the INC instruction increments a 32-bit counter in one shot, and the final JMP instruction requires the SMALL keyword to specify that OldTimer contains a seg:off address.*

```
PROC        TickHandler

PUSH        AX
PUSH        DX

CALL        OpenRAM             ; enable writes

INC         [CS:TickCtr]        ; TickCtr is 32 bits wide

CALL        CloseRAM            ; disable writes

POP         DX
POP         AX

JMP         SMALL [CS:OldTimer.DWORD]

ENDP        TickHandler
```

Although it's not written down anywhere, BIOS requires that you restore at least DS and ES in addition to CS, which is loaded by the final RETF. I save and restore all the segment registers even though the actual requirements surely depend on which BIOS you're using.

FDB EXT also marks a departure from the code you've seen so far: notice that I'm now using the FS and GS segment registers that appear only in '386 and higher CPUs. As a result, this code will not run on 8088 or 80286 systems. I don't include any tests for the CPU type, as I assume we're all adults around here. *Don't* try it on your old clunker PC just to see what happens.. .it won't work!

Yes, FDB EXT could be written to work on any 80x86 CPU, but it's time to start using hardware that's been around since 1985. OK?

CAPTURING INTERRUPTS

The remainder of FDBEXT's initialization code captures the BIOS timer and nonmaskable interrupt vectors. This is standard code that you've seen before, so I won't waste space on the listings.

Listing 3 shows the timer interrupt handler. The Firmware Development Board's RAM is normally write-protected, so each handler must enable the RAM before updating the variables. The CS segment stored in the interrupt vector is the same as the DS value set up in Listing 2, so the INC instruction can "reach" TickCtr using CS without having to save, load, use, and restore DS.

Because the RAM write-enable bit shares the same port as the watchdog timer bit, it is easy to have the OpenRAM and CloseRAM toggle the watchdog on each BIOS timer tick. Measuring the bit's active time shows that the interrupt handler requires about 30µs.

As I mentioned in the last column, it's generally not a good idea to toggle a watchdog from a timer interrupt because the main routine can crash without affecting the timer tick. However, this will keep the watchdog at bay while loading a *big* program from diskette. The mainline code can always capture the timer tick and implement my favored method after it starts running.

Enabling '386 assembly mode has some interesting side effects. The TickCtr variable is a double word, but the assembler uses the 32-bit version of INC to update it in one instruction. The JMP at the end of the routine passes control to the previous interrupt handler, but you must specify SMALL to tell the assembler that the vector represents a Seg:Off value instead of a 32-bit LARGE offset in the current segment.

I like that sound.. .even in real mode!

FAILING POWER

The BIOS extension responds to power failures by write-protecting the RAM and spinning in a safe loop. While writing this code I uncovered a nasty bug-NM1 glitches. While these shouldn't pose a problem in most systems, it's worth thinking about them if you're using the MAX691.

The Firmware Development Board includes a trimpot to adjust the voltage on the MAX691's PFI pin. The correct setting activates the Power Fail Output when the supply voltage falls near the system's lower tolerance limit; say -5% on a ±10% system. The remaining 5 % gives you enough time to shut the system down before the supply goes out of tolerance.

In small systems, the MAX691 is the only source of nonmaskable interrupts, but, as I described in the last column, many parts of a PC contribute to the NMI signal. Our handler must examine the board's power failure status and chain to the previous NMI handler. Only when the PFO bit is low can the handler shut down the system.

Here's the problem: if the supply voltage falls slowly enough, a small supply glitch that would normally be well within tolerance can trigger the PFI comparator and generate a nonmaskable interrupt. By the time the CPU responds to the NMI and checks the PFO status bit, however, the glitch is long gone. You can simulate this by adjusting the PFI trimpot very slowly.

Because none of the NMI sources are active, the default BIOS handler gets control. Guess what? On my system, the default handler disables further NMIs from the ISA –IOCHCK signal! So when the power really fails, the NMI handler never gets control.

If the FDB is the only source of IOCHCK interrupts in your system, your interrupt handler can check the status bit in port 0x61 to verify that the NMI came from the bus. Because that status bit is latched when IOCHCK goes active it does not go off when the glitch vanishes.

However, if you have several I/O cards that can produce IOCHCK interrupts, the situation is a little messier. A 1-nF capacitor on the PFI trimpot's wiper filters the glitches with a 10-µs time constant. On my system that made the trimpot tease-proof, but you should evaluate it to make sure it does not delay the interrupt too much during a real power failure on your system.

An alternative approach would be some hysteresis on the PFI pin. Because PFO switches low as PFI drops, a resistor between the two pins will yank PFI down and prevent the end of the glitch from restoring PFO.. .assuming the comparator's propagation time doesn't glitch it the other way!

In any event, one of the conditional assembly options shown in Listing 4 is a timing loop that starts on the first NMI. It display the loop count on the LEDs until the next NMI, at which time it locks up the system. You can use that code [which is in EXTTEST.BIN on the BBS] to see

Listing 4—*The NMI handler normally shuts down the system in response to a power failure. If the supply voltage falls very slowly (or if you tease the FDB's trimpot), you can get a glitch on NMI that vanishes by the time this handler gets control. The code shown here includes an optional section that displays the elapsed time from the first NMI to the next, then locks up the system. If this is a problem in your system, the code can also lock up in response to an NMI caused by the ISA –IOCHCK input*

```
            PROC    NMIHandler

            PUSH    AX
            PUSH    DX

            CALL    OpenRAM         ; enable writes
            INC     [CS:NMICtr]     ; record this NMI
            CALL    CloseRAM        ; disable writes

            IF      COUNT_NMI       ; show delay?

            MOV     CX,0            ; set up the counter
@Wait:
            MOV     DX,LED_ADDR     ; show the loop counter in binary
            MOV     AX,CX
            NOT     AX
            OUT     DX,AX

            IF      USE_IOCHCK

            IN      AL,SYS_CTLS     ; look at IOCHCK flag
            TEST    AL,IOCHCK
@Kaput:     JNZ     SHORT @@Kaput   ; lock up when it goes high
```

(continued)

Listing *4-continued*

```
            ELSE

            MOV     DX,STAT_ADDR    ; check power status
            IN      AX,DX
            TEST    AX,PWR_GOOD
@Kaput:     JZ      SHORT @@Kaput   ; lock up when status goes low

            ENDIF

            INC     CX
            JMP     @@Wait

            ELSE

            MOV     AX,1510h        ; show ni (more or less)
            MOV     DX,LED_ADDR
            NOT     AX
            OUT     DX,AX

            IF      USE_IOCHCK

            IN      AL,SYS_CTLS     ; look at IOCHCK flag
            TEST    AL,IOCHCK
            JNZ     SHORT @Lockup ; lock up when it goes high
            ELSE

            MOV     DX,STAT_ADDR    ; check power status
            IN      AX,DX
```

(continued)

how this problem looks on your system.

The CPU disables all interrupts within the NMI handler, so the watchdog timer isn't updated in the final lockup loop. The MAX691 will time out and reset the system about 1.6 seconds after the second NMI occurs.

RESETS AND THE WORST HACK

The Original AT's designers had a problem. They needed a way to get their new 80286 CPU back to real mode even though the chip had no way to shut off its protected mode enable bit. The 80826 emerged from hardware reset in real mode, but once the program entered protected mode there was no way back. Their solution is a testament to engineering ingenuity.

The AT included an 8042 microcontroller to handle a variety of tasks that were done with discrete logic in the Original PC. The designers added a command to the 8042's repertoire that toggled the 80286

CPU's reset line active for about 6 µs. Blam.. .back to real mode!

But the BIOS normally clears the system RAM and runs power-on diagnostics immediately after a hardware reset, which is not quite what they wanted. So they reserved a byte at address 0x0 **F** in the real-time clock's battery-backed CMOS RAM to indicate the reason for the shutdown.

Before the BIOS gets too far, it asks the keyboard controller why the system was reset. If the controller says that it executed a reset command (as opposed to a power-on or front-panel reset), the BIOS reads the shutdown reason code. If that byte indicates a protected-to-real mode transition, the BIOS branches directly back to the mode switch routine.

The only reason you think it's a kludge is that you didn't design it. It's really a clean, general, and useful way around an otherwise insurmountable hardware limitation. Remember: you don't get paid if the system doesn't work!

Intel got the message loud and clear: starting with the 80386 all their CPUs enter and exit protected mode at the flip of a bit. By now, though a considerable body of software uses the '286 method, so you can buy hyperthyroid keyboard controllers with special fast-path hardware logic to recognize and speed up the reset command. I kid you not.

The shutdown reason code can select one of several different routines after a reset. Most of them are not suited for civilian use, but one may come in handy in certain desperate situations. I'll show how to use it, you figure out when it's appropriate. Fair enough?

If the shutdown reason code is 0x0A, the BIOS vectors through the value stored at address 0040:0067. Because the system RAM isn't affected by the brief shutdown, you can regain control immediately after a hardware reset. Of course, all the registers except CS:IP are lost, so there are a few minor details I'll leave as an exercise.

Listing 5a shows FDBEXT's rudimentary restart routine, which simply increments a counter and sends another reset command to the keyboard controller. The BIOS clears the shutdown reason code before branching to the routine, so it treats the second reset as a complete power-on reset.

Listing *4-continued*

```
            TEST    AX,PWR_GOOD
            JZ      SHORT @@Lockup ; zero = power NOT good...

            ENDIF
            ENDIF

            POP     DX
            POP     AX
            JMP     SMALL [CS:OldNMI.DWORD]
@@Lockup:
            MOV     AX,08080h       ; both decimal points
            CALL    ShowBits

@@Stall:    JMP     @@Stall

            ENDP    NMIHandler
```

Although the restart handler is in nonvolatile RAM (it must be there when it's needed!), it seems that **F D B E X T** cannot load the vector. At least on my system, the restart vector changes after **F D B E X T** exits, although the shutdown reason code does not. So **FDB E XT** puts the address it would have used in a spot that **EXTTEST** knows about.

Listing 5b shows the code from **E X TT E S T that** transfers the vector from nonvolatile RAM to address 0040:0067 and sets the shutdown reason code. Later, in response to a keyboard command, **E X TT E ST** simply executes the following instruction to reset the system: outp(0x64,0xFE). The BIOS then executes the code in Listing 5a, goes through a second reset with all the normal power-on tests, and reloads **EXTTEST** from diskette. That's all there is to it!

So if anybody asks you about the worst hack in PC-dom, you can say you've been there and done that. Be sure to tell me if you put it to good use!

A CAUTIONARY TALE

I have often advised you to read the data sheets carefully. Generally I do a lot of reading before starting a project, but once in a while, well.. This tale shows that hell hath no fury like that of an unjustified assumption.

Those of long memory will recall the series of articles on INKnet back in late 1989 ("A Network for Distributed Control," *Circuit Cellar INK,* issues 10-12). To summarize, the networks used a 19.2-kbps RS-485 serial link to connect up to 32 nodes. I wrote a monitor program for IBM ATs that acted as a console for the 8052-BASIC nodes and displayed network status information. All in all, a neat project.

Because the design point was an 8-MHz AT from IBM, running a network at 1920 bytes/second posed some interesting challenges. Because an AT runs at about 1 MIPS, there are only about 500 instructions between each byte. If you get distracted for a millisecond or so you will lose data.

Each transmitted byte actually generates two interrupts because the RS-485 network echoes data back to the receiver. The first interrupt occurs when the transmitter buffer goes empty and the second, very shortly thereafter, blinks on when the receiver buffer fills with the same character. The elapsed time varies, but it can be as little as one bit-time, or about 50 µs.

Because the two interrupts occur so close together, I polled for interrupts at the end of the handler to eliminate the lengthy interrupt exit-and-entry overhead if the byte was ready. My 'scope showed that this worked quite well: most of the time, each character produced only one interrupt.

About a year later, though, some customers reported sporadic problems with network errors that we simply couldn't duplicate. Some of the problems were due to cabling, some to terminations, others to severe noise.. .but there was a very small minority of customers with everything set up right and everything still going wrong.

The problems seemed more severe on faster machines. Finally, this year one customer installed the software on his new 66-MHz 486DX2 and reported that it failed in a matter of minutes. Ah ha! The bug must be related to CPU speed, because, in this case, we'd eliminated everything else.

I set up a test network on my then-new 33-MHz '386SX (yes, the same one I'm using for these embedded '386SX projects], activated the trace outputs built into all my code, hitched up the logic analyzer, and waited to see what happened. After a long wait, the TSR got jammed in an "impossible" state.

Although my system wasn't as fast as the latest 486 CPU, the logic analyzer showed that each outbound character generally produced two separate interrupts. Progress had eliminated the need for my interrupt polling trick and I was glad to get rid of it. However, very rarely, the second interrupt (the one caused by the receiver buffer) was suspiciously long.

I modified the TSR code to produce trace outputs for each possible interrupt source and discovered that the "long" interrupts were caused by a change in the modem status register. That was peculiar, as the TSR did not enable MSR interrupts.. .and the Interrupt ID Register should not report a disabled interrupt.

Essentially, all PCs use National Semiconductor 8250, 16450, or 16550 serial interface chips or an LSI chip that works just like them. I pored over the data sheets in search of something I'd missed three years ago. What could cause an invalid IIR! I assumed that my code was at fault, as genuine hardware problems are very, very few and far between.

In the 8250 family, transmitter interrupts are cleared when you read the IIR or write a new character, but receiver interrupts are cleared only when you read the pending byte. You would expect, as I did, that the Interrupt ID Register is updated almost immediately. You would be *almost* correct.

The IIR reports the highest-priority pending interrupt, but bit 0 is a summary status flag that, when zero, means "there is at least one interrupt active." My code reads the IIR and uses it as an index into a decoding table. Only two interrupts can occur in the INKnet TSR, but, being a belt and suspenders type, my table has all possible entries. That saved my skin!

Upon close scrutiny, the 16450 data sheet reveals two key timings. The summary bit is updated within 250 ns of reading the IIR for transmitter interrupts. Should the receiver cause the interrupt, however, it may take up to 1 µs to flip the summary bit after reading the character. The interrupt request output pin has the same timings, so the IIR bit must be wired to the output driver rather than the actual input bits on the chip.

As Sherlock puts it, "When you have eliminated the impossible, whatever remains, *however improbable*, must be the truth."

A sufficiently fast CPU can respond to the interrupt, read the IIR, branch to the receiver handler, read and process the byte, and check the IIR again *before the summary bit changes*. Because the receiver interrupt bit is cleared almost immediately, the IIR is invalid.

As you might guess, an all-zero IIR indicates a modem status interrupt.

Listing **5a**—This code in FDBEXT.ASM gains control after the keyboard controller blips the CPU's reset line. The BIOS checks the shutdown reason code at address OF in the real-time clock's CMOS RAM; if that value is OA it vectors through the address stored at 0040:0067, which EXTTEST aims at this routine.

```
            PROC    StartupHandler

            CALL    OpenRAM             ; enable writes

            INC     [CS:StartupCtr]     ; record another startup

            CALL    CloseRAM            ; disable writes

            MOV     AX,00580h           ; show r. on the LEDs
            CALL    ShowBits

            MOV     AL,OFEh             ; tell kbd controller
            OUT     KEY_CMD,AL          ;... to blip the reset line

@Stall:     JMP     @Stall

            ENDP    StartupHandler
```

Listing **5b**—*This code from EXTTEST.C loads the vector and sets the shutdown reason code into the real-time clock's CMOS RAM. Note that the vector is not in the interrupt table because it is neither a hardware nor a software interrupt.*

```
printf("Changing startup vector from %04x:%04x to %04x:%04x...\n",
     peekw(0x0040,0x0069), peekw(0x0040,0x0067),
     peekw(EXTSEG,STARTUP_SEG), peekw(EXTSEG,STARTUP_OFF));
pokew(0x0040, 0x0067, peekw(EXTSEG,STARTUP_OFF));
pokew(0x0040, 0x0069, peekw(EXTSEG,STARTUP_SEG));
putstr(" setting shutdown code...\n");
disable();
outp(RTC_ADDR, 0x0f);          /* aim at shutdown reason code */
outp(RTC_DATA, 0x0A);          /* vector through 0040:0067    */
enable();
putstr(" done\n");
```

I checked my references again to see if anyone else knew about this. The only hint was in Mark Nelson's ***Serial Communications: A C++ Developer's Guide,*** published in ***1992*** (well after I needed it). In 8250 *Oddities* he states:

"...One annoying bug found in both the original National Semiconductor chips as well as some clone chips is the false modem status interrupt. The IIR can report a modem status interrupt when none has occurred. This could easily lead to trouble with the ISR code.. ."

Indeed!

The serial port in my 33-MHz '386SX is just one corner of an LSI chip, but ISA compatibility barnacles dictate exactly how it must work. In this case, the barnacles require new silicon to precisely duplicate the same old bugs! Aren't standards great?

There are no good fixes for this, so I used a time-honored kludge: a delay loop. The code measures the CPU speed when it installs the TSR and sets up a delay loop that occupies at least a microsecond. After each receiver interrupt, it stalls long enough to ensure that the IIR interrupt summary flag is valid. Not pretty, but it works.

The moral of this story is twofold: RTFM first, then build trace outputs into your code so you can see what's going on. But any regular reader of this column knows that already, right?

RELEASE NOTES

The BBS files this month include the BOOTSECT loader to start "application" programs from diskette and the special LOAD **EXT** loader to put a BIOS extension into the FDB's nonvolatile storage. You also get **FDB EXT** and EXTTEST so you can see how the whole process works. The comments explain how to load and run the code.

Next month I'll add a small character LCD to the Firmware Development Board so you can display messages without a serial port. I'll also give each FDB a unique serial number to make up for stealting the keyboard lock switch. This may smell like copy protection, but in the embedded systems world there are some excellent reasons to make sure your code runs only on the right machines! ❑

Ed ***Nisley, as Nisley Micro Engineering, makes small computers do amazing things. He's also a member of the Computer Applications*** *Journal's* ***engineering staff.*** You ***may reach him on CompuServe at*** *74065,1363* ***or through the Circuit Cellar BBS.***

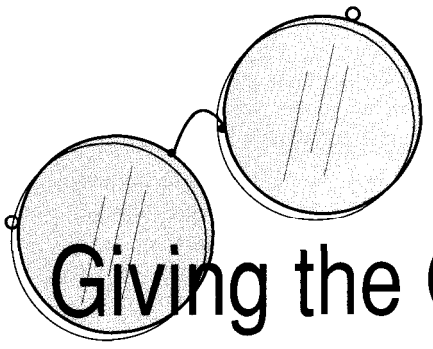
SOURCE

If you've gotten this far in the project you should have no trouble finding a pushbutton switch. Pure Unobtainium has the complete Firmware Development Board schematic, as well as selected parts. Write for a catalog:

Pure Unobtanium
13 109 Old Creedmoor Rd.
Raleigh NC 276 **13**
Phone/fax: (919) 676-4525

IRS

410 Very Useful
411 Moderately Useful
412 Not Useful



Jeff Bachiochi

Giving the Optical Card Reader Improved Vision

Concluding his optical ID card reader project, Jeff adds a Hewlett-Packard optics package to improve the reader's resolution and adds a dedicated processor for some network-ready intelligence.

Over the past month, my eyesight seems to have degraded to the point where I thought a checkup was in order. I couldn't understand how my vision could fail so rapidly. I tossed around thoughts trying to pinpoint a cause. Could it be a change in diet? No, I am still having my weekly pizza fix. How about the weather? Well, it is hotter and I am swimming more, but Crystal Lake doesn't have chlorine.

It is the obvious we tend to overlook. The glasses I wear have round lenses. The lens "crafter" was kind enough to put a slight notch on the lens where it should line up with the frame joint. While cleaning my eyewear this morning, I noticed the lens was gradually rotating in the frame. After realigning the notches, I could see clearly again.

RECAP

Last month, I discussed optical coding techniques and some methods for printing optical swipe codes using standard character graphics. I showed you how a two-byte character string could be encoded on a small credit card, and how it could be read using inexpensive microsensors. The bit-checked code is capable of 2^{16} (65536) different combinations.

INCREASING RESOLUTION

Higher quality optics can increase the focusing power for both the transmitter and receiver, which enables detection of objects with a finer resolution. The more concentrated the area of illumination is at the

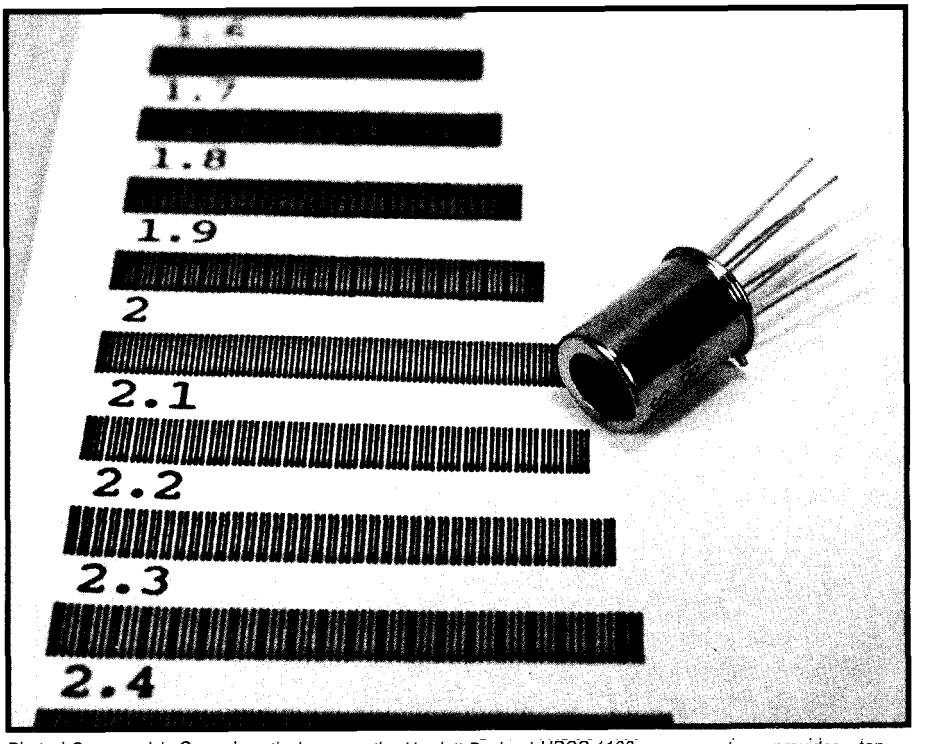


Photo I-Compared to Omron's optical sensor, the Hewlett-Packard HBCS-1100 sensor package provides a ten-fold improvement in detection of a barcode.

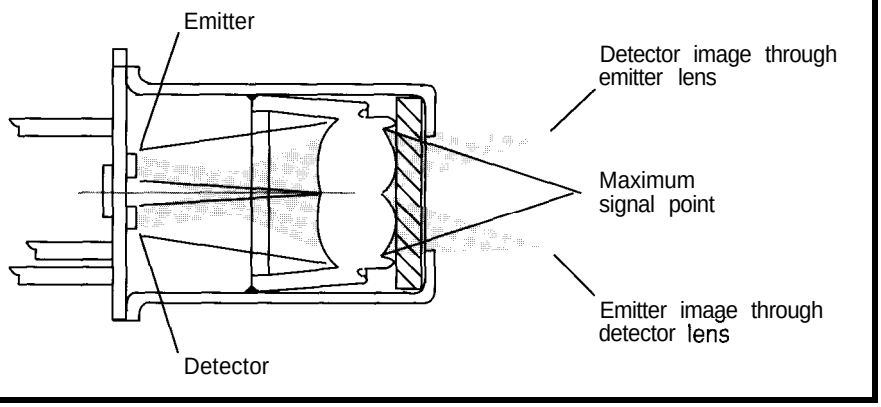


Figure 1--The *Hewlett-Packard HBCS-1100 focuses a single 0.190-mm spot 4 mm in front of the bifurcated aspheric lens* assembly.

focal point, the smaller a nonreflective spot can be that prevents reflective detection. The HBCS-1100 from Hewlett-Packard uses a bifurcated aspheric lens to image the active areas of a visible LED and photodetector to a single 0.190-mm spot, that is focused 4 mm in front of the lens. This is neatly packaged in an 8-pin TO-5 can. HP's sensor is $27.00 (single piece) as opposed to Omron's at $4, but the specs are also in a different league. (see Photo and Figure 1)

If you assume a barcode that uses a 0.2-mm line and 0.2-mm space, the HP sensor should be able to read about 60 bits/inch (about 8 bytes, or 4 bit-checked bytes/inch). This indicates that a factor of 10 improvement is realized over Omron's device. The HBCS-1100 includes an internal transistor which can (optionally) be used as a high-gain amplifier to the photodiode.

WHAT GOOD IS FINER RESOLUTION IF YOU CAN'T PRINT A LABEL?

A printer's character graphics are limited to 80 character columns per 8.5 inches (or 132 character columns in condensed mode]. This comes up to about 15 characters per inch (132 characters/8.5 inches) which is just about equivalent to 1 byte of bit-checked data/inch. This is about the minimum size we could detect with the inexpensive microsensors using a slit mask. So how can we print these small swipe codes?

The vertical line character is the thinnest possible line which can be printed. The problem is not so much with printing this character, but in printing each character with a smaller-than-normal character-to-character width. This calls for some special carriage control. The LaserJet is capable of carriage movements down to 1/300 of an inch. Listing 1 was used to print multiple characters with a varying amount of back carriage movement between characters. Using visual inspection, I chose a carriage movement that left a space approximately equal to the width of the character to assure detection. This spacing measures about 0.3 mm/line-space width.

It may not sound like a major improvement, and compared to magnetic densities it's a joke, however we now have 2^{48} different possible combinations within the same density. Not bad for the home-brewed approach.

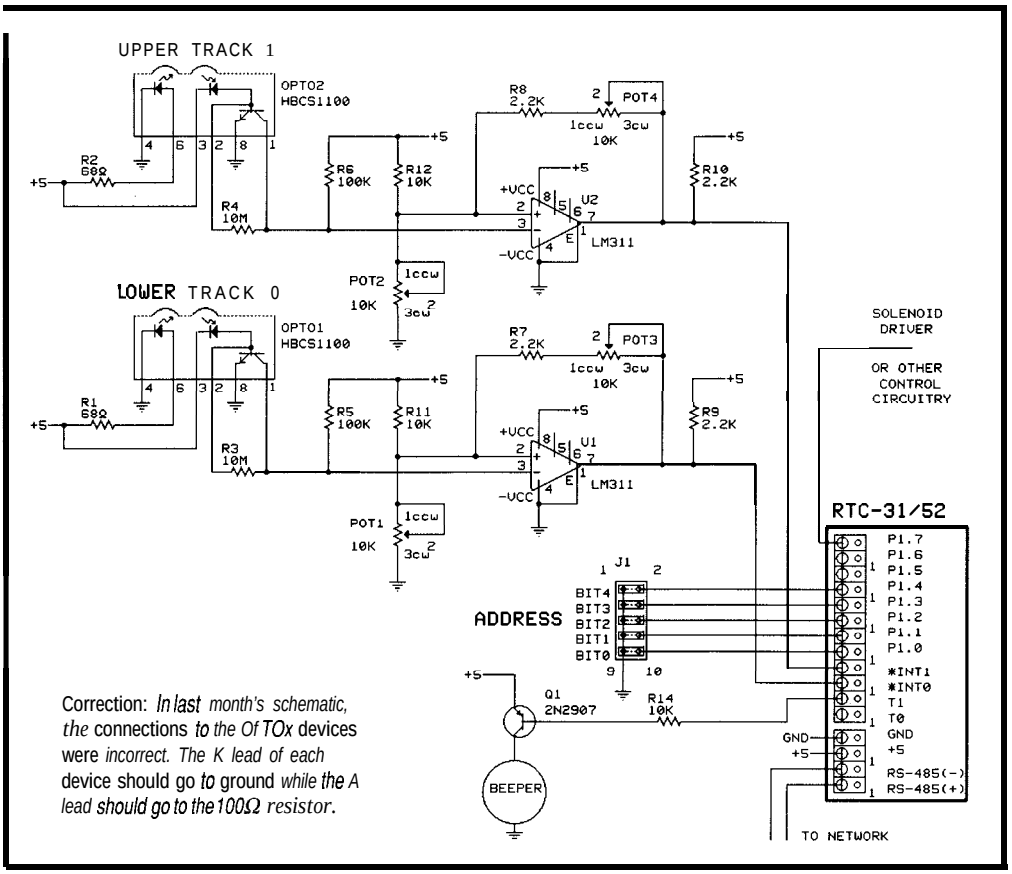
CUT DOWN A FEW NOTCHES

It isn't too often you hear anyone advertising their failures. If I didn't feel there was lesson here, I would bury the following experiences deep within my research paperwork. But, since I think there is something to be learned here, I will humble myself and share it with you all. The project I had in mind was a black box which would read optical swipe cards, and respond either by beeping an error tone or by performing some other function. The functions that would be performed might include storing the card sequence; comparing the card sequence to a list of acceptable sequences, unlocking a door, notifying a remote computer, allowing the remote to "OK" the card sequence, and so forth.

My plan was to use a 68HC705K1 or a PIC processor to act as a networked node of the swipe reader

Listing 1—*Using spacing tricks with the HP LaserJet, it's possible to achieve a 0.3-mm/line-space width without resorting to using graphics mode.*

```
10 N = 12
20 WIDTH "LPT1:",255
30 LPRINT CHR$(27) + "&k2.9H"
40 DIM H(N-1), S1$(N-1), S2$(N-1)
50 M$ = CHR$(&H7C): S$ = " "
60 FOR X = 0 TO N-1
70 PRINT "Enter hex number ";X;: INPUT "(0-15)?", H(X)
80 S1$(X) = "": S2$(X) = ""
90 NEXT X
100 SS1$ = M$+M$+M$+M$+S$: SS2$ = S$+S$+S$+S$+M$
110 SE1$ = M$+S$+S$+S$+S$: SE28 = S$+M$+M$+M$+M$
120 FOR X = 0 TO N-1
130 V = H(X)
140 FOR Z = 3 TO 0 STEP -1
150 IF (V AND (2^Z)) = 2^Z THEN S1$(X) = S1$(X)+M$+S$
                               S2$(X) = S2$(X)+S$+M$
                          ELSE S1$(X) = S1$(X)+S$+M$:
                               S2$(X) = S2$(X)+M$+S$
160 NEXT Z
170 NEXT X
180 LPRINT SS1$;:FOR X=0 TO N-1  LPRINT S1$(X);:NEXT X: LPRINT SE1$
190 LPRINT SS2$;:FOR X=0 TO N-1  LPRINT S2$(X);:NEXT X: LPRINT SE28
200 LPRINT
210 LPRINT CHR$(27);"&k12H"
220 END
```



Correction: *In last month's schematic, the* connections *to the Of TOx* devices were *incorrect. The K lead of each* device should go *to* ground *while the A lead should go to the 100Ω resistor.*

Figure 2—*Comparators are used to clean up the raw* signal from *the* optical *sensors. The* squared-off signals then drive interrupt lines on *the RTC52. The* single-board *computer adds the* intelligence *necessary to make a network-based,* stand-alone unit.

system. My hardware requirements for this node were a built-in UART or multiple timers and external interrupts. None of the small processors have UARTs, which means bit-banging a serial port. That's fine, but I can't lose interrupts from the optical tracks, either. I decided to break this into two independent operations. The first would be responsible for reading the swipe tracks and verifying a valid read. The second would handle the serial network and talk to an EEPROM which would hold a list of acceptable data sequences.

I spent a few days writing optical data code and the interprocessor nybble port (for communications between processors), I spent a few more days writing network code and debugging the interprocessor nybble port, then a day or so on EEPROM routines. I linked the routines together and received an "attempt to use nonexistent memory." Do I add a third processor in parallel to handle the EEPROM routines? This was starting to smell a bit like it was turning into a parallel processing article. Good material, but not what I started out to accomplish.

So this is what it's like when you can't fit ten pounds into a five-pound bag. It always fit before. Oh well, from now on I'll severely overestimate the size of my code, just to be safe!

NOW WHAT?

Let's take a moment and review the requirements again. I needed a UART, some nonvolatile memory, a couple of interrupts, and some form of network hardware. Well, this list of requirements could be met with the RTC31/52. Nothing lowers the cost of a design like using the hardware over and over for a variety of tasks. I guess the wide range of applications is what gives single-board computers their universal value. For those of you who are curious or are not familiar with the RTC3 1/52, see "From the Bench" in the April/May 1989 issue of the *Computer Applications Journal* (issue #8) for the schematic.

Figure 2 shows the schematic for the swipe circuit and how it connects to the RTC31/52. The RTC52's built-in BASIC interpreter is used for the foreground task of analyzing the network "command strings." If a command string with its own address (00–30) is recognized, the appropriate response is given back to the network. Supported commands are:

Query-a list of all completed swipe data sequences

List-a numbered list of all acceptable data sequences

Clear-clear the list of acceptable data sequences

Add-add an acceptable data sequence

Delete-delete a particular data sequence

The original approach, using an EEPROM, had room for 64 two-byte entries. Using the RTC52 with a nonvolatile RAM increases this to thousands. Also, the original system could store only a single swipe. This meant any network master had to keep on its toes and be ready to jump at a moment's notice. Now, there is plenty of buffer space which takes a big burden off of any network master.

Since improved resolution allows more than the initial two bytes of data, the system can be configured to read two or more bytes of data from a single

	Swipe direction		Which bit has the actual data
1	forward	(MSB-LSB)	first bit
2	forward	(MSB-LSB)	complement bit
3	backward	(LSB-MSB)	first bit
4	backward	(LSB-MSB)	complement bit

Figure 3—*Depending on the* direction a card is *swiped and which bit has the* actual *data, four ways* exist *to* actually collect *the data.*

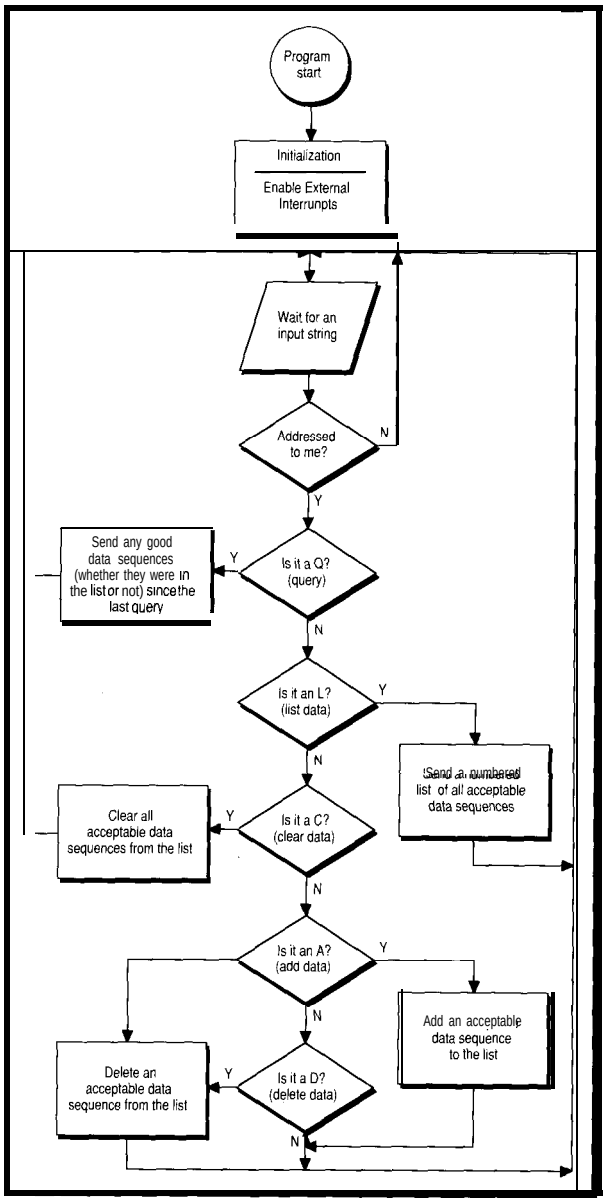


Figure 4a—*The BASIC foreground* task *analyzes the network* command *strings and acts on the supported command set.*

card. The number of stored sequences goes down proportionally to the sequence length, but there is still plenty of room to go around.

HIGH-PRIORITY TASKS

Two interrupt routines are written in machine language. The first is the T I ME RO overflow interrupt. This routine is responsible for identifying when a swipe is completed. If the overflow counter reaches its maximum count and an error or an uncompleted read has been flagged, then an error beeper is strobed and things are reset to await another swipe. If a good read was confirmed, the data sequence is transferred to the queue and a comparison is performed against all the acceptable data sequences in the list buffer. If a match is made, an output bit is strobed enabling a door latch (or other device). If no match is found, the error beeper is strobed and things are reset for the next pass.

The TIMER0 interrupt is enabled and the timer is cleared each time the external interrupt routine is entered, thus preventing a timeout. This routine can be entered by either one of the two external interrupts. EXTO is triggered by a mark on the lower track. EXT1 is triggered by a mark on the upper track. The data bit is set to a "0" or "1" by the corresponding interrupt.

Three steps are necessary to complete a good swipe. First, the start sequence needs to be recognized. Since all data bits are sent once, complemented, and sent again for confirmation, a start sequence cannot be confused with data. Second, a start sequence has at least three consecutive bits of the same logic state followed by one complemented bit. The last bit is used to flag the direction of the card swipe.

Third, the end sequence must be the complement of the start sequence but in the reverse order. Thus, the only two possible start and end sequences are:

111 O--/ /-l 000
or
0001—//—0111

The data is of a known length, which is a function of sensor resolution, printer resolution, and card size.

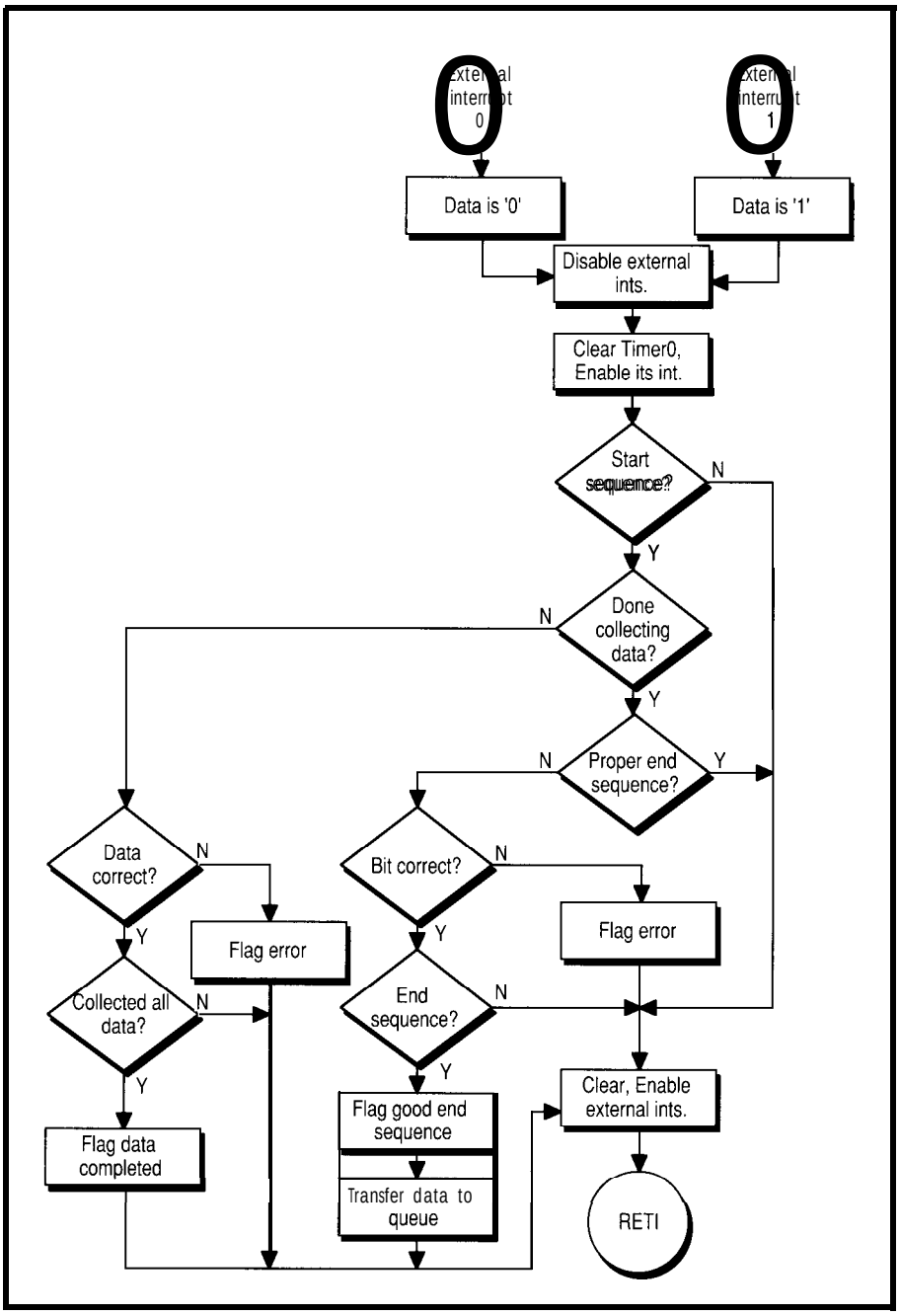


Figure 4b—*External interrupt 0 is triggered by a mark on the lower track (a 0 bit) and external interrupt 1 is triggered by a mark on the upper track (a 1 bit).*

The data is expected as "bit" and "complemented bit" to ensure data integrity. There are actually four ways data can be collected (Figure 3).

I use the first data bit when reading forward, and the complement when reading backward. When the data sequence is transferred to the queue, I reverse the sequence if the read direction was backward. Any data that falls outside of the expected logic state sets the error flag. You can see the algorithm for the program I wrote for the swipe card reader in Figure 4.

TIME STAMP

You could easily add local time stamping to each entry, or you could allow the remote master to time stamp query polls (providing it does polling in a timely fashion).

Stand-alone operation (meaning that no computer is necessary to load the acceptance list) could be added to the security door application. This could be accomplished by assigning a master "add" and "delete" card sequence to the program that would place the unit in a mode to alter the

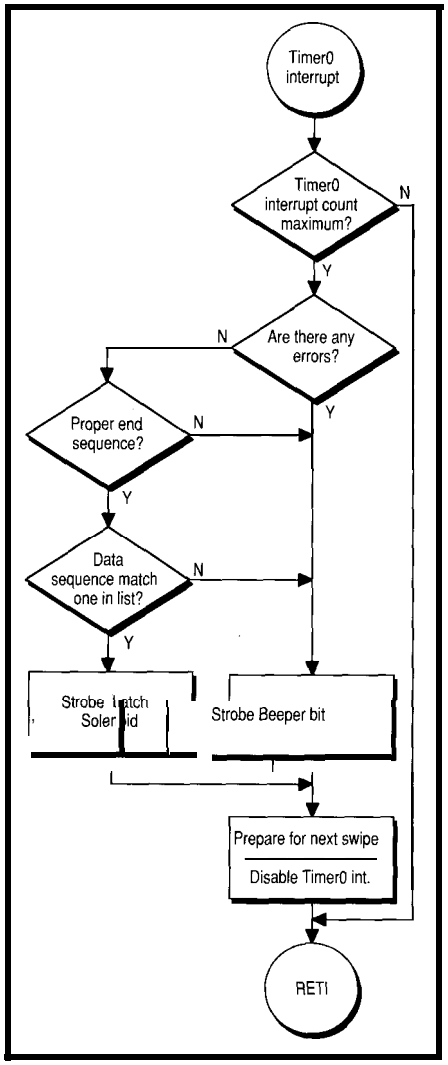


Figure 4c—*The TIMER0* interrupt routine is responsible for identifying when a swipe is *complete*.

acceptable data sequence list by just swiping cards through. That is, provided you are playing with a full deck.

Jeff Bachiochi (pronounced "BAH-key-AH-key") is an electrical engineer on the Computer Applications Journal's engineering staff. His background includes product design and manufacturing.

CONTACT

Hewlett-Packard
P.O. Box 10301
Palo Alto, CA 94303-0890

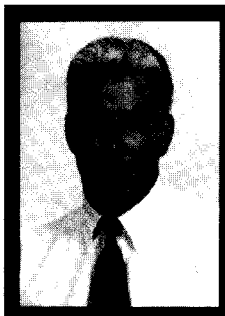
I R S

413 Very Useful
414 Moderately Useful
415 Not Useful

Penny-pinching OTPs

SILICON UPDATE

Tom Cantrell



When you're in a pinch for space and need to keep costs down, but must have more smarts than a simple gate array can provide, it's time to check out the latest round of tiny, inexpensive processors from Philips and Zilog.

I think I have discovered an unspoken tenet of the high-tech magazine business: The amount of breathless editorial coverage given to any new widget is inversely proportional to its relevance in real-world applications.

If you were to judge yourself by typical trade journal covers, you would be a Neanderthal (or an incurable Luddite) if you chose not to use the latest 32-bit SuperDuper Wonderchip.

Well, I just have to say, "Humbug to that noise!" Sure, these multimegagate wonderchips are useful-especially when it comes to replacing the "big iron" of old. But I've always felt that the biggest benefit of the silicon revolution is reserved for the low end. In the final analysis, it comes down to the difference between doing something that has already been done before (albeit less expensively) versus doing something that has never been done. This class of unique, new, "low end" products is only now possible because it can finally be done cheaply enough to make the exercise worthwhile.

For instance, I recently read that cars produced today have on average 70+ chips manufactured into them. Current trends indicate this number will increase. Admittedly, the main ECU (Engine Control Unit) is likely to be a rather high-tech device, but much of the really neat stuff (like antilock brakes and airbags) are made possible by mass-produced ICs whose main "feature" is that they are priced low enough to prevent terminal sticker shock.

So let's get some cheap thrills by taking a look at a couple of popular penny-pinching OTP (One Time Programmable, or EPROM in a no-window plastic package) controllers—the 87C750 from Philips and the Zilog Z86E08. These devices are shown in Photo 1.

LESS IS MORE

Starting with packaging, Photo 1 shows you won't have to fuss with fragile, high-pin-count, hard-to-assemble-with packages. The '750 is available in a 24-pin skinny-DIP (0.375 square inches) or a 28-pin PLCC (0.25 square inches). The 'E08 comes in an 18-pin DIP (0.28 square inches).

Let's move inside for an even closer look. Figures la and 1b show the

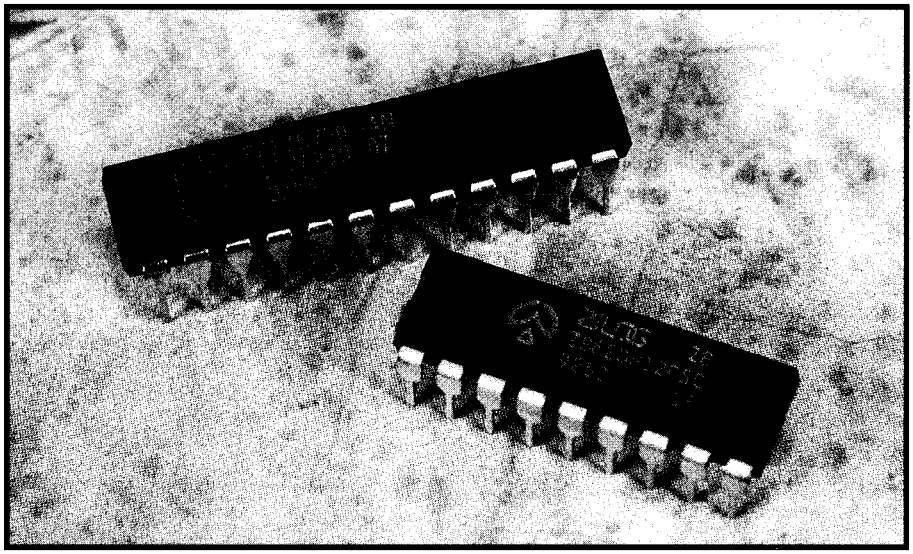


Photo I--Penny-pinching *OTPs from Philips and Zilog include RAM, PROM, and I/O all on one chip.*

block diagrams of these chips. With I/O functionality limited by the pin count, the 'E08 offers 14 I/O lines while the '750 manages to offer 19. Keeping efficiency in mind, notice how the 'E08 only needs four lines of overhead (V_{ss}, V_{cc}, XTAL1 and XTAL2), while the '750 adds a fifth (RST). Though the 28-pin PLCC version of the '750 doesn't use the extra four pins provided by the package, it still packs the largest number of I/O lines in the smallest board space.

Both chips are miserly when it comes to doling out memory, so put your C compilers away for now. The 'E08 offers a cozy 2K bytes of OTP memory and 144 bytes of RAM while the '750 is lean-and-mean with only 1K byte of OTP memory and 64 bytes of RAM.

Other than this trim offering of memory, little more than timers and counters are deemed worthy enough to receive any of the scarce transistors on these chips. The '750 provides a single 16-bit timer/counter while the 'E08 opts for twin 8-bit units, each with a 6-bit prescaler.

Boy, these simple chips sure make life easy for us writers-see you next month.

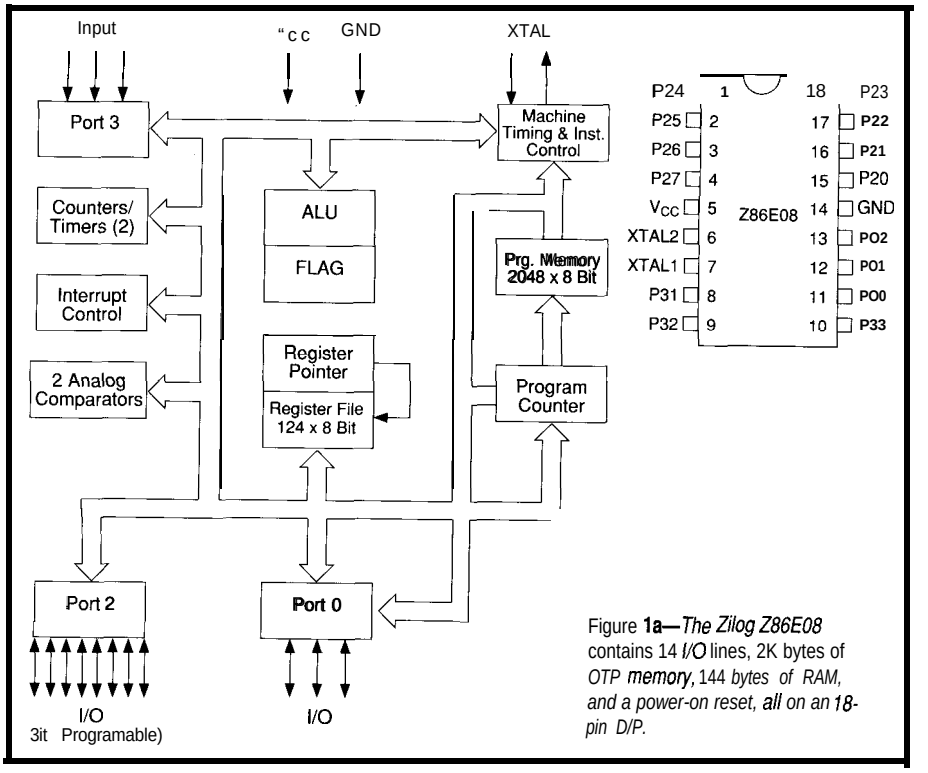
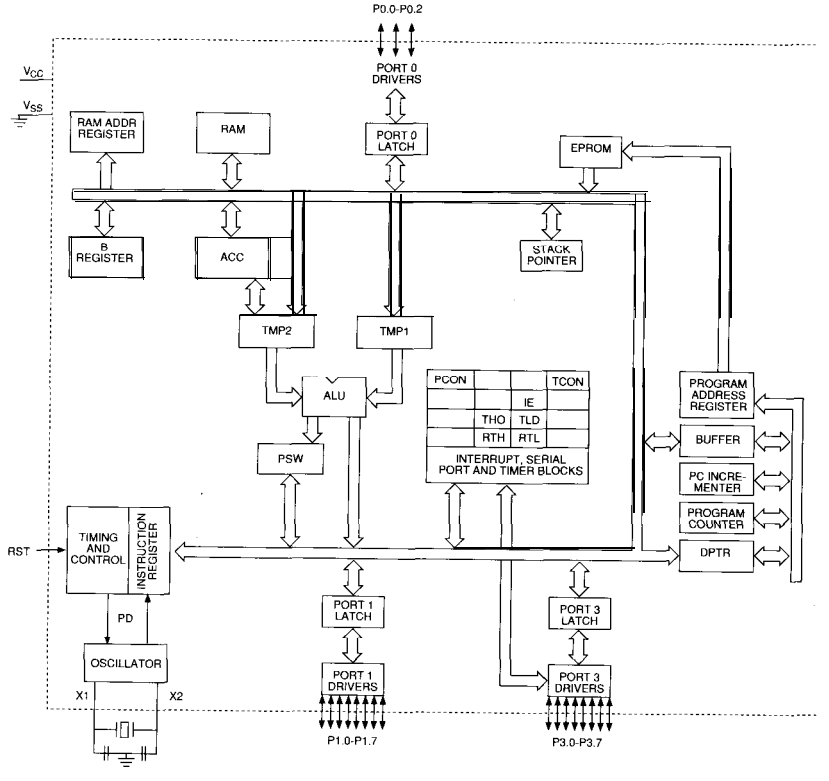


Figure **1a**—*The Zilog Z86E08* contains 14 *I/O* lines, 2K bytes of *OTP memory,* 144 *bytes of RAM, and a power-on reset, all on an 18-pin D/P.*

NOT SO FAST

Since our beloved editor didn't fall for my little ploy, let's take a closer look at the features of each chip while keeping a watchful eye out for any style-crimping gotchas that might be hiding to bite you later.

"Not so fast" is also the way to describe these CPUs so we can avoid long-winded performance analysis and "architecture wars." At entry-level clock speeds-12 MHz for the 'E08 and 16 MHz for the '750—both chips toddle along at about 0.5-l MIPS.



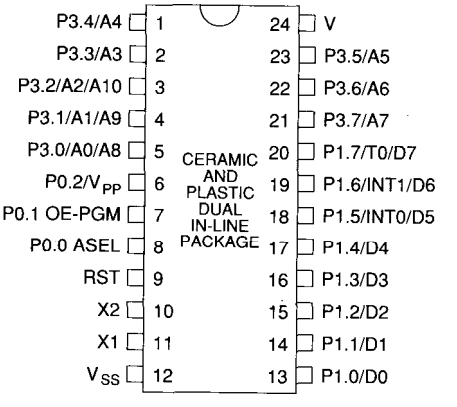


Figure **1b**—*The Philips 87C750 has 19 I/O fines, 1K byte of OTP memory, and 64 bytes of RAM in a 24-pin skinny-D/P or a 28-pin PLCC package*

Sure, there are some minor differences, but nothing worth quibbling about. The 'E08 makes up for a slower clock with a 2-stage pipeline and a clean instruction set. On the other hand, even though the '750 suffers from accumulator constriction disease (bottleneckitis), it does feature some pretty fast (3-µs) multiplies and divides.

Both CPUs utilize separate code and data spaces, with special instructions (MOVC for the '750 and LDC for the 'E08) for reading data from the program space (useful for lookup tables, constants, etc.). I suspect code density for the two chips is similar, averaging something less than 2 bytes/instruction. This limits program size to around 500 lines for the '750, and 1000 lines for the 'E08.

Figures 2a and 2b show the respective memory maps for the 'E08 and '750. Though the 'E08 data sheet trumpets 144 bytes of RAM, only 124 bytes can be used for general purposes. The rest of the space is dedicated to I/O and control-related functions. Meanwhile, the 64-byte RAM space of the '75O'is divided into four banks of eight registers, 16 bytes of bit- or byte-addressable RAM, and 16 bytes of regular (i.e., byte addressable only) RAM. The dozen or so '750 I/O and control SFRs (Special Function Registers) aren't counted as part of the 64 bytes.

Location		Identifiers
255	Stack Pointer (Bits 7-0)	SPL
254	General Purpose Register	GPR
253	Register Pointer	RP
252	Program Control Flags	FLAGS
251	Interrupt Mask Register	HMH
250	Interrupt Request Register	IRQ
249	Interupt Priority Register	IPR
246	Ports 0-1 Mode	P01M
247	Port 3 Mode	P3M
246	Port 2 Mode	P2M
245	T0 Prescaler	PREO
244	Timer/Counter 0	TO
243	T1 Prescaler	PRE1
242	Timer/Counter 1	T1
241	Timer Mode	TMR
127	Not Implemented	
4	General Purpose Registers	
3	Port 3	P3
2	Port 2	P2
1	Reserved	P1
0	Port 0	P0

Figure 2a—*The 'E08* RAM layout dedicates 124 *bytes* to general-purpose *registers and the rest to I/O and control-related functions.*

I/OUs

The nineteen I/O lines of the '750 are divided into one %-bit port (port 0), and two 8-bit ports (ports 1 and 2). I'll award them extra brownie points for the fact that every pin on every port is individually definable as being an input or output. Each of the pins of port 0 is distinguished with open-collector drivers while the pins of ports 1 and 2 are TTL compatible, and include internal pull-ups. If the alternate functions (*INTO and ● INTl) or the timer input (TO) are used, then the width of port 1 is reduced since each of these requires a bit from port 1.

The 14 I/O lines of the 'E08 are allocated to a pair of 3-bit ports (ports 0 and 3) and an 8-bit port (port 2). Notice that this part has no port 1. This device is more restrictive than the '750, because only the 8-bit port (port 2) has bit-programmable pins that can be used as input or output. Port 0 is globally (all three bits together) programmable as input or output. Port 3 is hard wired as input only.

The 'E08 does manage to sneak in an extra feature [shown in Figure 3). Port 3 of the 'E08 offers two analog comparators (P3 1 and P32}, and a reference input {P33}. However, these same pins can also function as the external interrupt inputs (IRQO-IRQ3). Unfortunately, the analog or digital function can only be selected for the entire port and not on a bit-by-bit basis. Though not shown in the figure, pin 1 of port 3 can be used for the optional (and only) external timer input. This is the case whether the port is configured as an analog or a digital port. Finally, bit 7 of port 2 serves double duty as an optional input that can be used to bring the CPU out of STOP (low power) mode.

One note of caution-make sure to check the drive capacity levels of any I/O pin to verify compatibility with whatever you're connecting it to. This is especially important, since both of these chips use CMOS drivers/receivers. Generally, the devices are TTL compatible, though it appears the 'E08 needs pull-ups on inputs driven by TTL.

PART TIMERS

Though it's a little more cramped on the I/O front, the 'E08 does pull ahead slightly because it offers two timers as opposed to the '750's single timer unit. Otherwise, the timing and counting capabilities of these two CPUs are quite similar.

The 'E08 (Figure 4a) features 8-bit counters and 6-bit prescalers for a total resolution of 14 bits. The input clock is derived from the crystal frequency [the clock runs at OSC/8}, so its range is from 666 ns to 10.9 ms. From the figure you can see that timer I can optionally utilize one of the bits of port 3 as an external clock input. This bit can also serve as a gate/trigger for the internal clock. Each timer has initial value registers that can optionally be called into play for automatic reload. Each timer can generate an interrupt which (when considered along with the four possible external inputs) brings the total number of interrupts handled by the 'E08 to six.

The timer in a '750 (Figure 4b) is basically the same as an 805 1 timer operating in mode 2 (automatic reload], except its resolution is extended to 16 bits. This clock is also derived from the crystal (it ticks away at a rate of OSC/12}. The resolution of this timer (750 ns) is slightly lower than the resolution of the timer in the 'E08, but the 16-bit range of the '750 extends all the way up to 49.2 ms. Like the 'E08, the '750 can sacrifice an I/O line (bit 7 of port 1) for use as an external timer input. Unlike the 'E08, gating external signals uses a second pin of the port. The good news is that this allows gating of an external timer input, something the 'E08 can't do. The bad news is that the line used for gating is the same one used by INTO', leaving a lonely INT1 • as the only external interrupt source. By the way, the '750 abandons the 805 l's two-level programmable interrupt priority scheme in favor of a fixed priority—INTO * , timer, INT1 • .

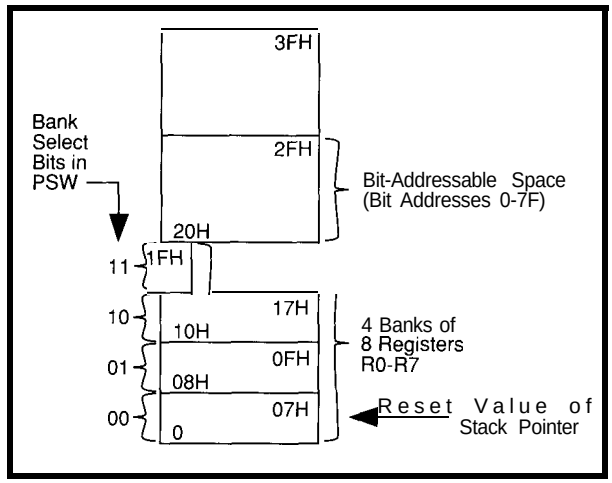


Figure **2b**—*The* '750 RAM *layout shows four banks of eight registers and 16* bytes of bit-addressable RAM.

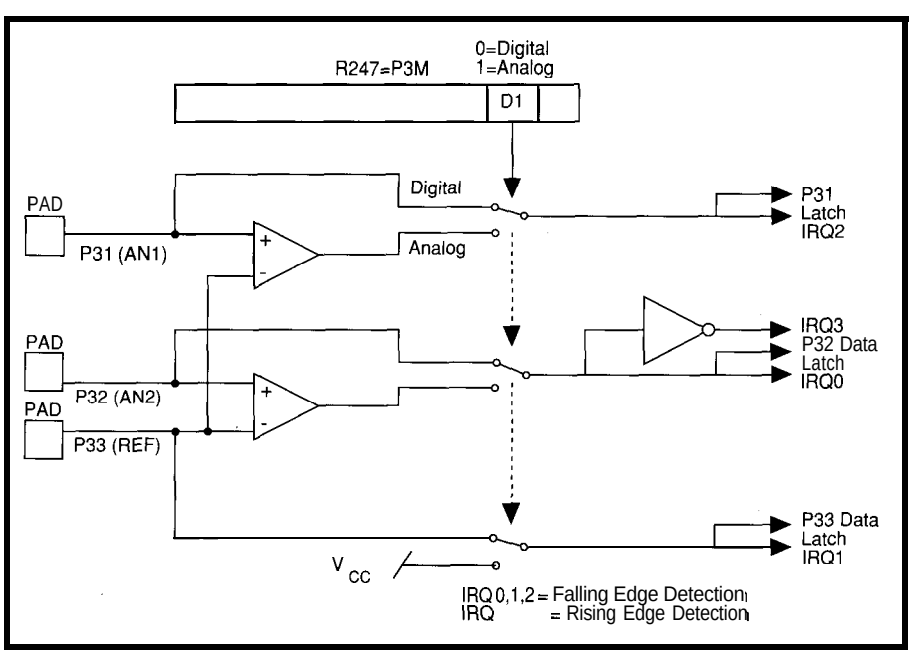


Figure 3-One *of the Z86E08's* unique *features is a* pair of *analog* comparators. *The* analog inputs share pins *with the regular digital pork and interrupt inputs.*

GREEN MACHINES

One of the primary methods of economizing in embedded systems is the reduction of power consumption. Both of these chips are positively Scrooge-like when it comes to running up the electric bill.

The '750 typically consumes only 12 mA when it is running at 16 MHz (with V_{cc}=4.5–5.5 V) and, as usual for CMOS, the power consumption

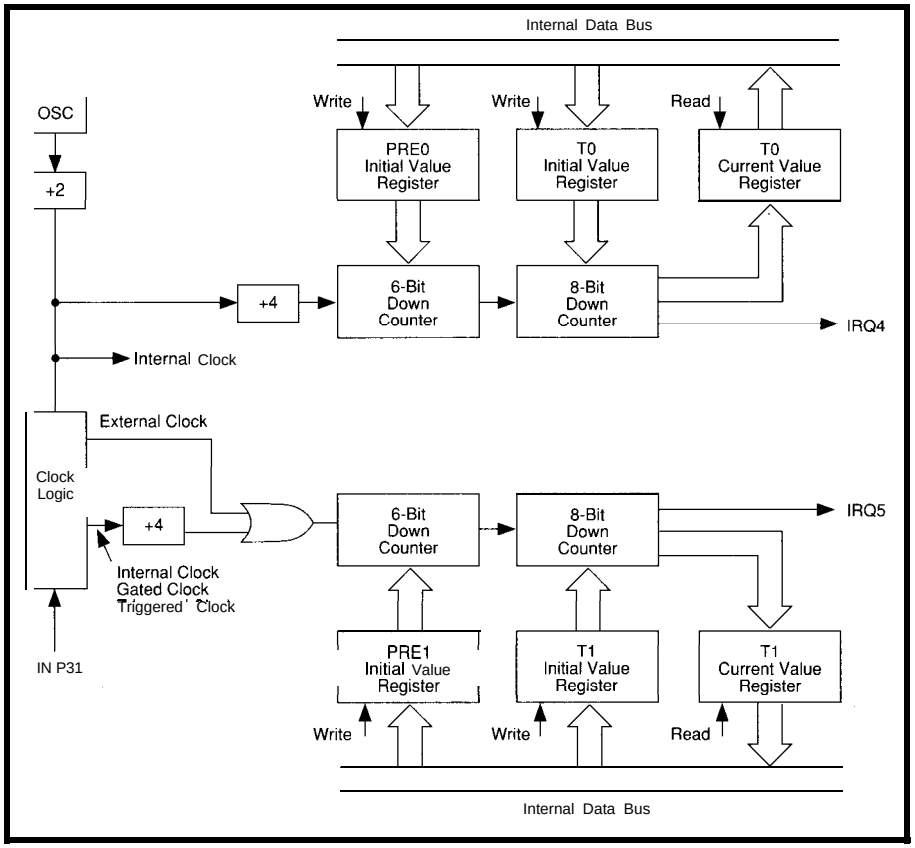


Figure 4a—*The 'E08* timers *have 14-bit* resolution by using *8-bit counters and 6-bit prescalers. Timer 1* can also use an external clock ***input.***

declines linearly with the clock rate. Philips has added their own low-power modes-IDLE and POWER DOWN—to the '51 architecture. IDLE mode defines a semiawake state in which all CPU activities stop, but the state of all other peripherals on the chip (I/O, RAM, SFRs, etc.) remain intact. Power consumption during IDLE mode is typically 2 mA, and waking up the chip is quick and easy in response to any enabled interrupt. POWER DOWN mode is comatose indeed, consuming a maximum of 50 µA and preserving only on-chip RAM. POWER DOWN is only recoverable through a reset.

The 'E08 is even more power conscious and, much to its favor, it is specified to operate at both 3 and 5 volts. Even when this device is running at 12 MHz and powered by 5 volts, its active I,, is typically 9 mA. Power consumption drops to 3.6 mA at this clock speed when the chip is powered by a 3-volt supply. The low-power modes of the 'E08 (HALT and STOP) function very much like the IDLE and POWER DOWN modes of the '750. However, the 'E08 does offer you the choice of using a port bit (bit 7 of port 2) for STOP mode recovery. This has fewer side effects (such as initializing I/O) than a recovery by reset, and makes using the lowest power (only 10 µA) mode easier.

The 'E08 racks up a few more points by integrating a power-on reset delay (no reset pin needed), a 50-ms watchdog timer (which can be enabled or disabled during low-power HALT mode), and built-in, low-voltage ("brownout") protection. The 'E08 even includes a low EM1 mode in which output power and slew rate are reduced to minimize RF pollution.

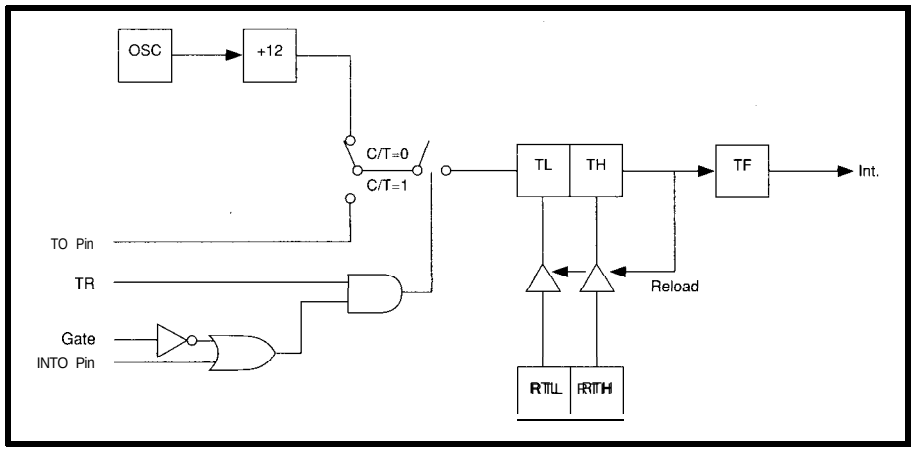


Figure 4b—*The '750 timer* is basically the same as an 8051 timer in mode *2,* but ifs resolution is extended to 16 bits.

PRICE IS RIGHT

Likely as not, one feature or another-such as the hardware math of the '750 or the brownout protection of the 'E08—will prove to be the compelling factor for certain applications. Whichever chip you choose, at $3.50 for the 'E08 and $4.50 for the '750 (lk, plastic DIP, 0-70°C) you're getting a lot of micro for your money. And at this low cost, you can afford to lash one of these devices to almost anything.

Tom** Cantrell **has been an engineer in Silicon Valley for more than ten years working on chip, board, and systems design and marketing. He can be reached at (510) 657-0264 or by fax at (510) 657-5441.

CONTACT

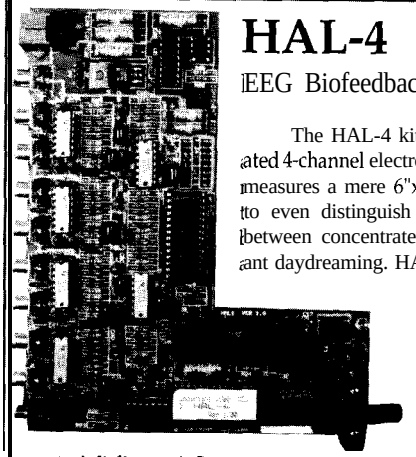
Philips Semiconductors
811 East Arques Ave.
P.O. Box 3409
Sunnyvale, CA 94088-3409
(408) 991-3737

Zilog, Inc.
2 10 Hacienda Ave.
Campbell, CA 95008-6609
(408) 370-8000

I R S

416 Very Useful
417 Moderately Useful
418 Not Useful

CIRCUIT CELLAR KITS



HAL-4

EEG Biofeedback Brainwave Analyzer

The HAL-4 kit is a complete battery-operated 4-channel electroencephalograph (EEG) which measures a mere 6"x7". HAL is sensitive enough to even distinguish different conscious states—between concentrated mental activity and pleasant daydreaming. HAL gathers all relevent alpha, beta, and theta brainwave signals within the range of 4-20 Hz and presents it in a serial digitized format that can be easily recorded or analyzed.

HAL's operation is straighforward. It samples four channels of analog brainwave data 64 times per second and transmits this digitized data serially to a PC at 4800 bps. There, using a Fast Fourier Transform to determine frequency, amplitude, and phase components, the results are graphically displayed in real time for each side of the brain.

HAL-4 kit.....**$179.00** plus shipping

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

Sonar Ranging Experimenter's Kit
Targeting ◆ Ranging ◆ Machine Vision

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

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

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

CHECK OUT THE NEW CIRCUIT CELLAR
HOME CONTROL SYSTEM

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

Call and ask about the HCS *II*

To order the products shown or to receive a catalog, call: (203) 875-2751 or fax: (203) 872-2204

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

#133

Fast Code Relief!

- Supports 8051 and Dallas DS5000 families
- Program in easy industrial BASIC (similar to GW-BASIC)
- Artificial Intelligence engine assists you with setup tasks
- Ring buffered interrupt driven serial I/O
- True time-based and external interrupts
- Supports unlimited in-line assemby code
- Your program can coexist with BASIC-52
- Same day shipping and 30 day money-back guarantee
- Competent technical support

"A potent development combination..." EDN January 20, 1992

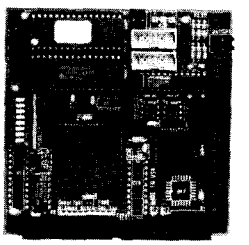
Systronix Inc.
555 South 300 East
Salt Lake City, Utah 84111
TEL: 801-534-1017
FAX: 801-534-1019

#134

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

MMT-188 EB

- *2 serial I/O ports*
- *3 programmable parallel I/O ports*
- *1 Meg RAM/ROM capable*
- *power fail detect interrupt and reset*
- *counter-timers*
- *watch dog timer*
- *expansion connector*



Only $191.00 *(Qty 100)*

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

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

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

Custom Work Welcome. Call or fax for complete data sheets

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

MIDWEST Micro-Tek
WE'RE SMALL BUT WE'RE POWERFUL

#135

Storage for the Masses

EMBEDDED TECHNIQUES

John Dybowski

Using a small microcontroller doesn't necessarily mean you're saddled with a small amount of memory. Using bank-switching techniques or even a special serial interface, lots of memory is well within your reach.

The term "mass storage" means different things to different people. Most computer-literate persons agree that it applies to some form of multimegabyte storage space. However, "mass storage" is different for many developers of embedded systems because these folks play in a different arena.

Most embedded systems are based on 8-bit controllers that don't have the capability to access significant amounts of memory. Even a 16-bit controller (such as an 80186) uses less-than-optimal methods to access megabytes of address space. Even this processor uses an architecture that is little more than a stylized paging scheme. Segmented memory architectures present enough headaches. If you add a banked memory scheme to this kind of system, you simply compound the problems associated with a segmented memory.

At first glance, a megabyte may seem like a lot of memory. But there are circumstances associated with the design tools used for 16-bit controllers that tend to nibble away at it. For instance, the development environments adopted for larger controllers often use compiled languages. Programs developed using these methods seem to have an inordinate hunger for substantial amounts memory.

Often, the memory space must be broken up to accommodate various families of memory components such as PROMs, RAMs, and flash devices. Any of these components may not be fully utilized, which even further reduces the available space. Large controllers often need memory expansion tricks as much as their smaller siblings.

There is an opposing force in embedded systems design. This force is the necessity to contain costs. This compulsion usually means that designs are done with cheap controllers. As a result, it is not uncommon to find instruments that use a $2.00 processor to drive $50.00 worth of memory. The situation is eased when the controller has a conventional parallel bus. This is especially true when on-chip I/O can handle the bank switching. Things get sticky when you try to pull it off with controllers that have very little I/O. As luck would have it, IC manufacturers come to the rescue and provide packaged answers to these concerns.

Now that the problem is understood, I'll describe ways to solve it. First I'll present a conventional bank-switching scheme that works with a standard bus architecture. Then I'll show you a way that operates using just three wires and can be adapted to a variety of controllers.

MEMORY BANKS

Banked memory schemes are structured in various ways. You can map memory into the I/O space (for processors having separate I/O instructions) or you can design banked RAM into the conventional memory space. The idea is to designate an address range as a memory page where different sections of physical memory can be swapped. Bank selection is usually done by manipulating signal lines that are used as an "address" by bank decoders.

The page size is made as large as possible to minimize how often bank changes must be done. I'm a big fan of dumb code anyway, and resist the temptation to make the code do any more than absolutely necessary. Circumstances sometimes change, so the trick is to avoid coding yourself into a corner. Although some malicious programmers take great delight in contriving diabolically complex sequences, I prefer to'allow code to function in a simple and mechanical manner unless there is some compelling reason to do otherwise. If code is

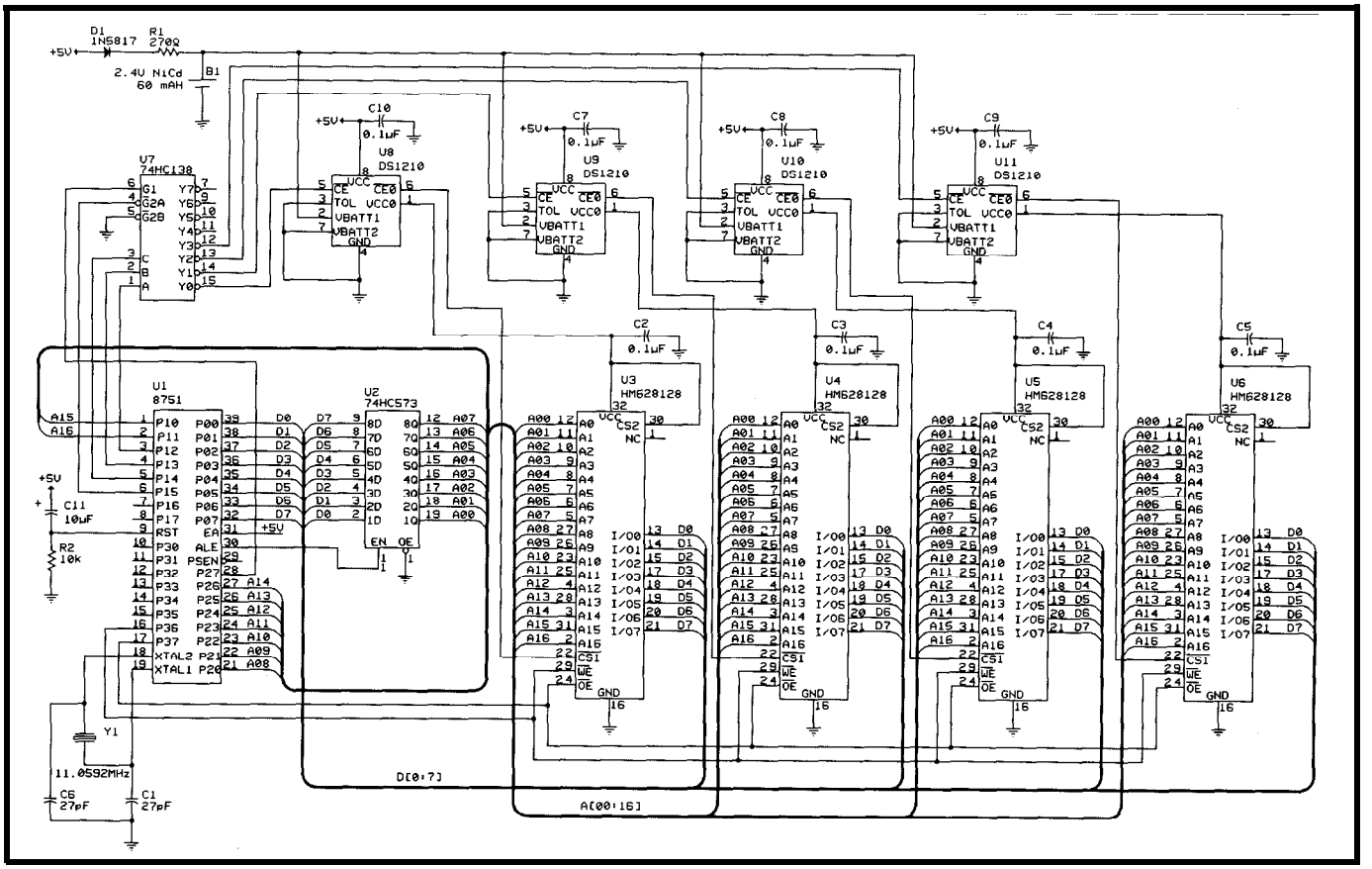


Figure 1—*By* using a *pair of output port bits, it's possible to select one of four 128K* banks *of memory, allowing a total of 512K of memory on an* 803 *I-based system. The DS1210 chips manage the backup* battery *which protects the RAM chips' contents.*

written properly, it's usually easy to add intelligence in upper layers if you have to be clever. In any case, making the bank size as large as possible helps.

An arrangement can be devised where several addresses are designated as address and data registers. The address registers describe the location to be accessed, and the size of the page is the size of the controller's data bus. This extreme situation does have the advantage of using only a few address locations. The disadvantage is you must explicitly set up the entire address for each access. The method you use depends on the processor, the way the system operates, and, most importantly, the way the data is used. If all you need to develop is a large buffer, or a circular queue, then you can take some license. If frequent random accesses are needed then the setup may be more restrictive.

BANKING ON THE 8031

Using the 803 1 to manage significant amounts of memory exemplifies the challenge of embedded instrument design. The problem with the 8031 is not its limited memory accessing capability. It's really not bad as controllers go when you consider its ability to access 64K of program memory and 64K of data memory. Its internal RAM, special function registers, and bit region offer several useful kinds of storage. The problem with the 8031 is that it is a "basic implementation" of a single-chip microcontroller and has serious restrictions in its external memory addressing modes. In spite of this, the 8031 is flourishing after fifteen years of service. New derivatives are being introduced regularly.

Although there are negatives, the 8031 does help you implement a banked memory system. One problem associated with bank switching is maintaining a section of RAM that is

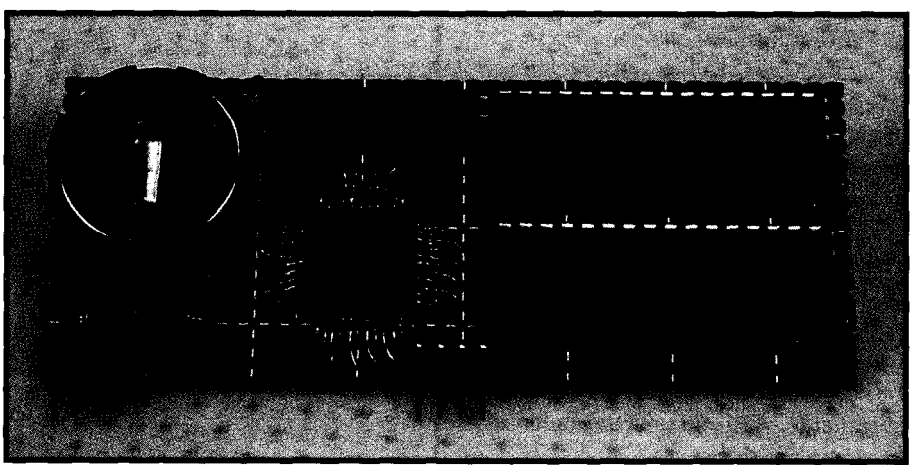


Photo I--The *DS1280 serial-to-bytewide converter can be used as the core of a serial-based memory module. Note that the chip is only available in a quad flat pack package.*

always available regardless of which bank is in service. An example of this requirement is the system RAM. The stack region, general-purpose buffers, and data areas required by interrupt service routines should always be active. If the system RAM were switched out when an interrupt occurred, mayhem would result. The other concern is the I/O required to perform bank switching. The I/O must be available regardless of which bank is active. It helps if this I/O is easily accessible to reduce the overhead of bank manipulations.

These problems can be circumvented with the **8031** since the stack is in internal RAM, and you might find enough internal RAM left over to satisfy the system's general purpose storage needs. The on-chip addressable I/O offers a way to access the pins to do the bank switching.

Now I'll demonstrate a bank-switching arrangement with the 803 1. Referring to Figure 1, you will see the memory system consists of four 128K SRAMs. The decoding is done by a '138 operating under firmware control via several on-chip I/O pins. Each RAM is battery backed using a NiCd battery composed of two cells and a Dallas Semiconductor DS1210 RAM controller. This arrangement provides a good match for these RAMs since the end-of-discharge voltage is 2 volts. This is the minimum voltage the RAMs require for data retention. Remember, CMOS SRAMs draw less current given a lower standby voltage, so it makes sense to keep the backup voltage low.

An important consideration in using RAM controllers with rechargeable batteries is the high end of the battery voltage. This voltage cannot exceed the VCC power-fail trip point or the part will never go into backup. Since the DS 12 10 is. designed to run off a lithium cell, the maximum battery voltage restriction makes sense. With two NiCd cells this is not a problem since the battery will never exceed 3 volts even when the cells are held in overcharge. With three cells you could be playing with fire.

The RAM area is partitioned into 32 pages. Each page is 32K deep and is

Listing I--Access *to the* banked region is *through a 20-bit pointer, which is supported using several routines.*

```
;THIS ROUTINE TRANSLATES THE 20 BIT LOGICAL ADDRESS TO WHAT
;THE HARDWARE REQUIRES TO ACCESS THE BULK RAM AREA.

;INPUT: R1 POINTS TO 20 BIT POINTER IN IRAM
;
SETUP_PTR:
        MOV     A,@R1           ;LSB
        MOV     DPL,A
        INC     R1
        MOV     A,@R1           ;NSB
        MOV     C,ACC.7
        SETB    ACC.7           ;ENSURE EXTERNAL ACCESS

        MOV     DPH,A
        MOV     P1.0,C          ;A15

        INC     R1
        MOV     A,@R1           ;MSB

        MOV     C,ACC.0
        MOV     P1.1,C          ;A16
        MOV     C,ACC.1
        MOV     P1.2,C          ;A17
        MOV     C,ACC.2
        MOV     P1.3,C          ;A18
        MOV     C,ACC.3
        MOV     P1.4,C          ;A19
        RET                     ;20 BIT ADDRESS IS SET
;
;THIS ROUTINE INCREMENTS THE 20 BIT LOGICAL POINTER.
;INPUT: R1 POINTS TO 20 BIT POINTER IN IRAM

INC_PTR:
        PUSH    ACC

        MOV     A,@R1
        INC     A
        MOV     @R1,A
        JNZ     IP1

        INC     R1
        MOV     A,@R1
        INC     A
        MOV     @R1,A
        JNZ     IP1

        INC     R1
        MOV     A,@R1
        INC     A
        ANL     A,#0FH
        MOV     @R1,A
IP1:    POP     ACC
        RET

;READ A BYTE FROM THE BULK RAM AREA
;INPUT: TEMP_RdPtr POINTS TO NEXT BYTE TO READ
;OUTPUT: TEMP_RdPtr IS INDEXED TO NEXT LOCATION

GET-BYTE:
        PUSH    DPL
        PUSH    DPH
        MOV     R1,#TEMP_RdPtr
        CALL    SETUP_PTR

        MOVX    A,@DPTR

        MOV     R1,#TEMP_RdPtr
```

(continued)

Listing 1-continued

```
        CALL    INC_PTR
        POP     DPH
        POP     DPL
        RET

;WRITE A BYTE TO THE BULK RAM AREA
;INPUT: TEMP_WrPtr POINTS TO NEXT STORAGE LOCATION
;OUTPUT: TEMP_WrPtr IS INDEXED TO NEXT LOCATION
;
PUT_BYTE:
        PUSH    DPL
        PUSH    DPH
        PUSH    ACC

        MOV     R1,#TEMP_WrPtr
        CALL    SETUP_PTR

        POP     ACC
        MOVX    @DPTR,A

        MOV     R1,#TEMP_WrPtr
        CALL    INC_PTR

        POP     DPH
        POP     DPL
        RET
```

located in the upper half of the data address space. This leaves the lower 32K available for a system area and general-purpose storage. The lower RAM can also be nonvolatile and can be used to hold control information and other variables necessary to operate the bank-switched area. With the 8031's separate program area, a full 64K of program memory is still available so you can write lazy code.

Access to the banked region is through a synthesized 20-bit pointer that is passed through a setup routine. Referring to Listing 1, you can see the lower 15 bits are placed into D PTR and the high-order bit is set to 1. This is done to displace the transfer region to the upper 32K of the address range. Bits 15 and 16 are moved to P1.0 and P1 .1 which drive the RAMs' address lines with the remaining three address bits presented to the '138 via P1.2, P1.3, and P1.4. Following this, the data transfer into or out of the RAM is

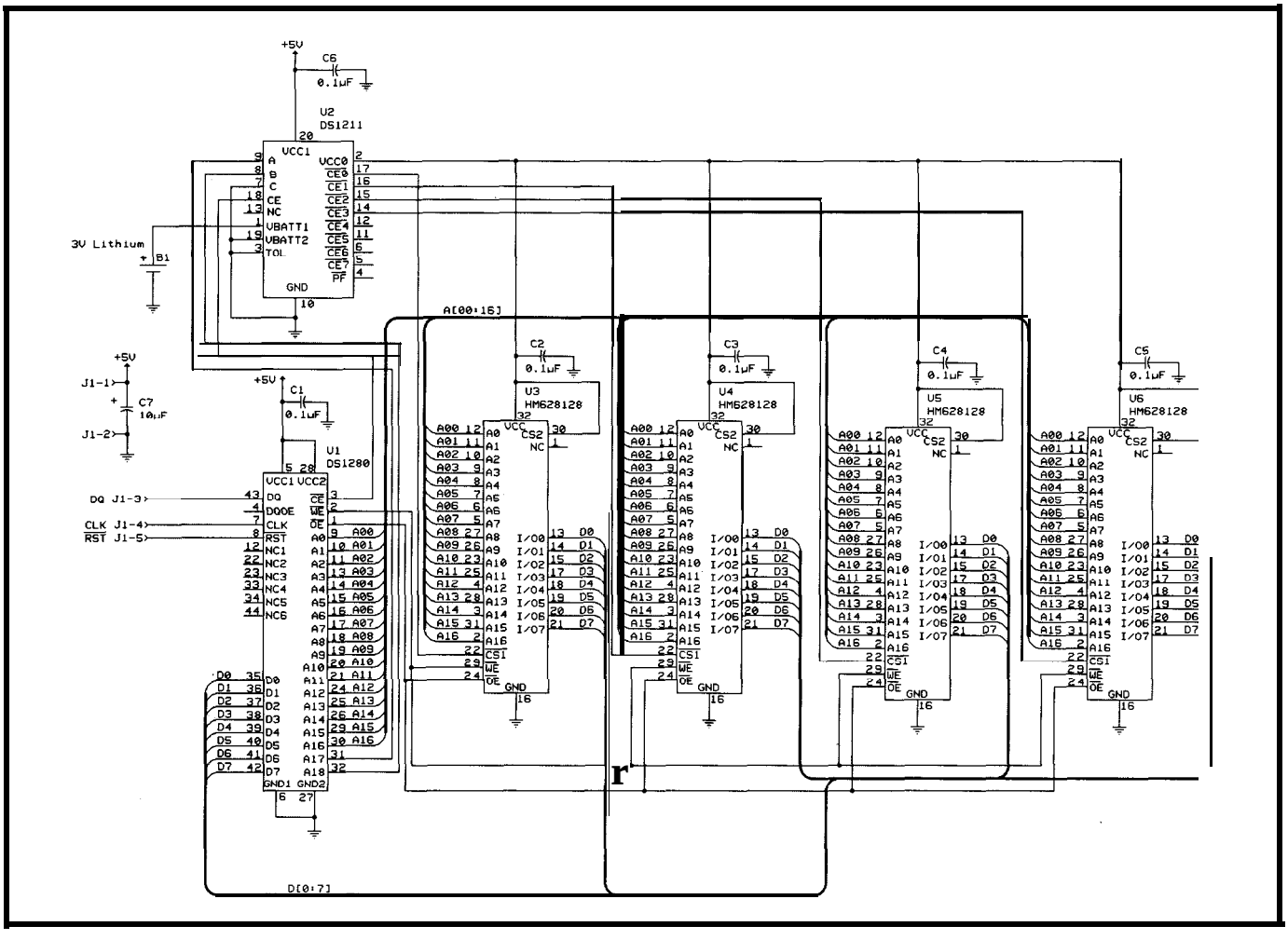


Figure P-The *DS1280 provides a clean interface between a simple three-wire* **serial** *interface and the mass of wires and connections* **associated** *with* **a large memory** *array.*

handled by second-level functions that execute a standard MO V X instruction. Although the logical address is a contiguous span that goes from 00000h to FFFFFh, the transfers occur at physical addresses that range from 8000h to FFFFh.

Also in Listing 1 are several support routines. Writing such constricted routines is more than an exercise in small thinking; the desired result is the flexibility such noncommittal function blocks offer. I used these basic services to craft a circular queue in a straightforward manner using conventional coding techniques.

SERIAL RAM

Effective as bank-switched RAM is, there are times when it pays to do the job with just a few wires. Reducing the interconnect burden is appealing when the RAM is on a separate card, where the reliability and cost of every connection is a prime concern, or when your controller does not have a bus architecture to begin with. In any case, serial interfaces are interesting and the throughput is often adequate. Contriving a serial RAM interface from scratch would involve a prohibitive amount of logic or the use of a dedicated microcontroller with its inevitable speed penalty. Due to the complexity involved in engineering a reliable serial interface, the end result would likely prove disappointing. I found a part that seemed like the answer to these problems. It took a little longer than I expected to get from here to there, though.

STEALTHSPECSHEETS

The Dallas DS1280 serial-to-bytewide converter looked like it would handle the serial conversion. Using a three-wire serial interface composed of a reset (\RST), a clock (CLK), and a bidirectional data (DQ) line, this IC is an example of another proprietary interface. I'm not a big fan of proprietary interfaces, but under the circumstances there's really not that much to complain about.

One interesting amenity in the DS1280 is a built-in CRC generator. In addition to checking the validity of commands, it can also be used by the

system controller to verify the data stream is written and read correctly. The inclusion of this safeguard should not be surprising since the DS1280 is the converter chip used in Dallas's CyberCard cartridges. In this application, the CRC is an important component that guards against interconnection errors due to intermittent contacts. It can also trap on errors when the cartridge is inserted and removed. I was ready to wire up my memory stick when I realized the DS1280 must be used primarily used by Dallas internally in its memory cartridge products. This realization came upon me after a cursory examination of the data sheet. With the problems I had figuring this part out, I came to the inescapable conclusion that nobody else could possibly be using it!

As they say, if you're going to screw something up, you might as well do a job of it. Since the DS1280 is only available in a quad flat pack (which meant hand wiring would be quite a chore], naturally one of my first concerns was in not making any

Listing 2—*The DS1280* memory interface is *supported by three routines.*

```
;Initialize the DS1280 controller and interface pins
;Set the I/O pins to their idle state
;Set the select bits to 0000000000000000b

Init_DS1280:
          setb CLK              ;set default clock level
          setb DQ               ;set default data level
          clr  RST              ;assert reset

;First read current (random) select bits

          mov  CRC,#0           ;initialize CRC value
          setb RST              ;release reset
          mov  a,#Read_Func
          call XByte            ;select read function
          mov  a,#0             ;send dummy address
          call XByte            ;low address bits
          mov  a,#0
          call XByte            ;mid address bits
          mov  a,#Select_Read   ;high addr and read command
          call XByte
          mov  a,#00000000b     ;send dummy select bits, CRC

          call XByte            ;low select bits
          mov  a,#00000000b
          call XByte            ;high select bits
          mov  a,CRC
          call XByte            ;protocol CRC value
```

(continued)

Listing *2-continued*

```
;now get the current select bits and save

          call RByte              ;read low select bits
          push acc
          call RByte              ;read high select bits
          push acc
          clr  RST                ;sequence is complete

;Now set the select bits to a known state

          setb RST                ;release reset
          mov  a,#Write_Func
          call XByte              ;select write function
          mov  a,#0               ;send dummy address
          call XByte              ;low address bits
          mov  a,#0
          call XByte              ;mid address bits
          mov  a,#Select_Write    ;high addr bi, write select cmd
          call XByte              ;send current select bits snd CRC

          pop  b                  ;retrieve high select bits
          pop  acc                ;retrieve low select bits
          call XByte
          mov  a,b
          call XByte
          mov  a,CRC
          call XByte              ;protocol CRC value
          mov  a,#00000000b       ;send the new select bits
          call XByte              ;low select bits
```

(continued)

wiring errors that would let the magic smoke out of the chip. My fears were not unfounded. Luckily, it didn't take me long to find out that there were problems with the spec sheet. The fact that a 44-pin quad flat pack has 11 pins on a side and the data sheet's diagram showed the nomenclature for 12 pin designations on one side was a dead giveaway. How's that for security against unauthorized use? After several calls to the factory, I had the correct pinouts but I must confess, my confidence was a shaken since there were other errors in the data sheet as well. Anyway, I'm happy to report I was able to get it working without blowing it up.

Referring to Figure 2, you can see how I devised a memory system around the DS1280. The DS1280 has a three-wire interface on the processor side and a standard address and data bus on the memory side. This memory interface provides the capacity of driving up to 512K of memory through 19 address lines, 8 bidirectional data lines, and an active-low chip select.

This chip also has a control signal (DQOE) that can enable a tristate driver when the DS 1280 is emitting data.

Nonvolatility is attained by using a Dallas DS1211 3-to-8 decoder/nonvolatizer IC and a lithium cell. The two high-order address lines feed the DS1211, which select one of the four RAM chips. The DS1280's chip select signal is an enable strobe to the DS 1211. The use of the enable ensures that all of the RAMs remain disabled, in a quiescent state, if a RAM access is not taking place. This keeps power consumption down. The remaining address lines are brought out to the RAM chips as are the data lines. Photo 1 is a picture of the result of my labors. Notice that this prototype has only two RAM chips for a total of 256K of storage-enough to prove the point.

SERIAL GYRATIONS

The DS1280 uses a 56-bit command sequence (including CRC) to initiate any action. Following the successful transfer of a command

Listing *2-continued*

```
          mov  a,#00000000b
          call XByte              ;high select bits
          clr  RST                ;sequence is complete
          ret

;Do burst write to RAM via DS1280
;input:    19 bit address is in r2/r3/r4 (r2 is lsb)
          dptr points to source buffer (xram)
          r0 contains byte count
;output:   acc=0 if write CRC is ok
;
Write_DS1280:
          mov  a,r0
          jz   BW_Exit            ;jump if nothing to write
          mov  CRC,#0             ;initialize CRC value
          setb RST                ;release reset
          mov  a,#Write_Func
          call XByte              ;select write function
          mov  a,r2               ;send address
          call XByte              ;low address bits
          mov  a,r3
          cal  XByte              ;mid address bits
          mov  a,r3
          anl  a,#111b            ;high address bits
          orl  a,#Burst_Write     ;and burst write command
          cal  XByte
          mov  a,#00000000b       ;send select bits
          cal  XByte              ;low select bits
          mov  a,#00000000b
```

(continued)

Listing *P-continued*

```
            call XByte              ;high select bits
            mov  a,CRC
            call XByte              ;protocol CRC value
BW_Loop:                            ;main data write loop
            movx a,@dptr
            inc  dptr
            call XByte              ;write data byte
            djnz rO.BW_Loop         ;verify CRC, result in acc
            call Check-CRC
BW_Exit:
            clr  RST                ;sequence is complete
            ret

;Do burst read from RAM via DS1280
;input:   19 bit address is in r2/r3/r4 (r2 is lsb)
         dptr points to destination buffer (xram)
         r0 contains byte count
;output:  ACC=0 if read CRC is ok
;
Read_DS1280:
            mov  a,r0
            jz   BR_Exit            ;jump if nothing to read

            mov  CRC,#0             ;initialize CRC value
            setb RST                ;release reset
            mov  a,#Read_Func
            call XByte              ;select read function

            mov  a,r2               ;send address
```

(continued)

sequence, you can perform burst (with automatic address incrementing) reads and writes to the RAM, read/writes of select bits, and reading the internal CRC register. Let me say a few words about the hardware signaling at the bit level before moving along.

A transition of RST from low to high clears the internal CRC register and enables communications for the DS1280. Dropping RST terminates any communications operation.

Data is clocked into the DS1280 on the rising edge of the CLK line, and data is clocked out of the DS1280 on the rising edge of the CLK line. All transfers are in octets with the LSB first, MSB last.

The actual communications protocol consists of several steps. The first byte sets up the general operation (i.e., read or write). The next three bytes are the address for the RAM operation and are transferred with the LSB first. They can be set to any value (usually 0) if the operation is not a RAM access. Only three bits of the high-order byte contain address

information, the remaining five bits denote a specific command.

Two bytes of selected bits are sent. These bits are intended to identify and differentiate multiple DS1280s. A CRC byte is computed over the entire command sequence and is also sent.

If the CRC is received correctly at the DS1280, the requested operation is carried out. If a CRC error is encountered, any further processing is terminated.

These steps must be followed for each access to the DS1280. The signaling overhead is made more palatable because you can transfer sizable chunks of data using bursts that autoincrement the DS1280's internal address generator at the conclusion of each byte. When using most general-purpose controllers, the DS1280 can handle the transfer rate about as fast as you can clock it. Now I will present some driver code for the DS1280 to better describe how it really works.

Listing *2-continued*

```
          call XByte              ;low address bits
          mov  a,r3
          call XByte              ;mid address bits
          mov  a,r3
          anl  a,#111b            ;high address bits
          orl  a,#Burst_Read      ;and burst read command
          call XByte

          mov  a,#00000000b       ;send select bits
          call XByte              ;low select bits
          mov  a,#00000000b
          call XByte              ;high select bits
          mov  a,CRC
          call XByte              ;protocol CRC value
BR_Loop:                          ;main read loop
          call RByte
          movx @dptr,a
          inc  dptr
          call XByte

          djnz r0,BW_Loop         ;verify CRC,  result in acc

          call  Check-CRC
BR_Exit:
          clr  RST                ;sequence is complete
          ret

          end
```

TEST DRIVING THE DS1280

Listing 2 shows three user-callable support routines that are used to operate the DS1280: Init_DS1280, Read_DS1280, and Write_DS1280. It's best to start with little pieces and work your way up. The first routines in this listing are defined to perform local support functions such as computing and verifying the CRC, and transferring bytes into and out of the DS1280. The remaining routines are accessible to higher layers and can do useful things now that the low-level stuff exists.

The initialization function is present since anything that operates with a serial interface usually needs some setup. What may be surprising is the amount of work this routine has to perform. After setting the I/O bits to their default state, the code drops through and sets the select bits to a known state. The select bits must be known in order to perform any useful RAM-related functions with the DS1280 since the DS1280 is not battery backed. As I mentioned, in any RAM operations, the transmitted select bits must match the select bits within the DS1280. It is possible to read the current select bits without knowing what they are, so I first read the select bits so I can write new select bits. Now I have the necessary information to tap the DS1280's capabilities.

Burst-read and burst-write routines follow the same general guidelines used in the initialization sequence. The inclusion of the read CRC command guarantees that the DS1280 has received or transmitted the same data that I thought it did. This success/failure information is communicated up to the higher-level processes where the appropriate actions can be taken should an error be detected.

SWITCH IN, SWITCH OUT

Developers of small embedded systems seem to be succumbing to the memory glut. In some cases, applications will benefit from this increase in storage capacity with added features and capabilities. Then there are those applications that would simply not be possible otherwise. In any case, if you play it right, the end result can be product differentiation and superiority. This is, after all, the ultimate goal.

Considerable amounts of memory can be accommodated by even the smallest of controllers. Using bank switching you can lash tons of RAM to any controller. With serial methods, even computational midgets can have the memory capacity of an Einstein. Next month, little memories. ❑

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

SOFTWARE

Software for this article is available from the Circuit Cellar BBS and on Software On Disk for this issue. Please see the end of "ConnecTime" in this issue for downloading and ordering information.

IRS

419 Very Useful
420 Moderately Useful
421 Not Useful

PATENT TALK by Russ Reiss

Back in the June issue I reported on a variety of patents in the field of communications. At that time, I mentioned that I had also located over 250 patents within the past three years dealing with spread spectrum, and that I would review them in a later issue. This exciting new field offers many radically new approaches to communications and other areas. It is something about which even a superficial knowledge would benefit engineers in many fields. While I cannot go into all the gory theoretical aspects of spread spectrum (SS) in this column, I hope to give you a nodding acquaintance with the subject by discussing a few of the more interesting applications gleaned from the patent search. You will see how SS fits well into this month's theme of "signal processing," for that's precisely what the entire concept is all about!

Simply stated, the idea behind SS communications is to spread the normally, somewhat narrow, spectrum of a signal over a much wider band of frequencies. In this way, the total power level used for transmission now appears as a minute amount of power at any one frequency, or over a narrow band of frequencies. The benefits of doing this are many-fold. The signal becomes nearly undetectable by a narrow-band receiver since there is so little energy within the narrow passband of such a receiver. While a wide-bandwidth receiver passes the entire signal, it is still undetectable unless the specific spreading code is known, because the spreading was done in a pseudorandom manner.

Either of two techniques are employed for spreading a signal. One is "frequency hopping" in which the carrier frequency of the emission is periodically or continuously altered in a (pseudo] random manner. The other is "direct sequence" SS in which a fast, pseudorandom, digital noise (PN for "pseudonoise") generator is used to modulate the carrier. The meaningful information is modulated onto the noise source. When the signal is spread over a wide band of frequencies, the fading that is due to multipath propagation is vastly reduced. So, too, is the effect of random noise and narrow-band interference on the decoded signal. Moreover, it is even possible for a number of stations (signals) to occupy the ***same*** frequencies at the ***same*** time, as one looks like "noise" to the other. This is true since they use different, and relatively uncorrelated, spreading codes.

When using binary coding, the RF carrier's amplitude, frequency, or phase is simply shifted between two states. Binary data, however, is not applied directly to the transmitter. Rather, each bit of the underlying signal (the data] is represented by ***many*** (typically hundreds or even thousands) of bits of the PN code. A zero (of the data) lets the PN code modulate the transmitter directly, while a one inverts the PN code. On the receiving end, it is necessary to generate a copy of the original PN code in order to synchronously demodulate the bit stream and recover the data. Since many (thousands of) bits of the PN code represent one data bit, there is a large integration process that takes place, and this represents the "processing gain" of the system-a ***real*** gain, just as though one increased the power of the transmitter, the sensitivity of the receiver, or used a better antenna.

With that as a thumbnail description of SS communication, let's look at some applications uncovered in the patent review. Abstract 1 proposes to use SS to provide a return link in a cable TV system. As you can see from the abstract,

1

Patent Number	4,912,721
Issue Date	1990 03 27
Inventor(s)	Pidgeon, Rezin E., Jr.; Zendt, Frederick T.; Thompson, Leo J.
State/Country	GA
Assignee	Scientific-Atlanta, Inc.
US References	3,750,022 3,886,538 3,943,447 4,002,843 4,475,208 4,494,138 4,586,078 4,635,274 4,706,284
Title	Cable television spread spectrum data transmission apparatus
Abstract	Apparatus for transmitting data spread across at least a portion of the bandwidth of a cable television channel comprises a carrier signal oscillator, a frequency divider, a pseudorandom sequence generator, and two exclusive OR gates. A first exclusive OR gate serves to spread a data signal across the pseudorandom noise sequence generator having a much higher chip rate than the bit rate of the data signal. The second exclusive OR gate modulates the spread spectrum data signal to a carrier frequency for transmission over the cable television channel. The apparatus may be applied for return path transmission in the 0–30-megahertz band which is highly susceptible to interference noise and provides approximately a 20-dB signal to interference ratio advantage over known data coding and transmission schemes. A microprocessor normally present in a cable television terminal may format data for transmission, control the spread spectrum modulation process, and control gain control circuitry for introducing an appropriate power level into the cable plant.

PATENT TALK

Patent Number	4,912,722
Issue Date	1990 03 27
Inventor(s)	Carlin, James W.
State/Country	NJ
Assignee	AT&T Bell Laboratories
US References	3,605,018 4,048,563 4,112,372 4,264,628 4,285,060 4,335,463 4,351,064 4,360,810 4,423,517 4,460,992 4,653,069 4,703,474 4,805,216 4,829,540
Title	Self-synchronous spread spectrum transmitter/receiver
Abstract	The present invention relates to a self-synchronous spread spectrum transmitter and an associated remote spread spectrum receiver which communicate with each other by the transmission of both (1) a spread spectrum Pseudo Noise Code (PNC) sequence signal, and (2) a combined PNC sequence plus the data information (PNC+data) signal. The two signals can be transmitted concurrently using either different frequency bands or on a quadrature carrier; or the two signals can be transmitted with a time offset between signals. At the receiver the PNC and the PNC+data signals are separately recovered for the case of the concurrent transmission techniques, and directly mixed to despread the received signal and recover the data signal at baseband. For the time offset technique, the delay provided in one of the time offset signals is again introduced to the previously delayed signal, and the delayed and undelayed signal portions of the received combined signal are directly mixed and then low pass filtered to recover the data signal. Such techniques eliminate the necessity for providing PNC acquisition and tracking circuits in associated spread spectrum transmitter and receiver combinations,

a rather conventional, direct-sequence, SS system uses a binary PN code generator to spread the data signal. Since all operations are performed on binary data up to the point of modulation onto the carrier, this is accomplished with a simple XOR circuit. Note the use of the term "chip" to refer to the bits of the PN code, reserving "bit" for the data stream. Here, as always, many chips are used to make up each data bit. This results in the "processing gain" when these chips are synchronously integrated at the receiver to recover the data stream. As pointed out, SS offers its many benefits to this application: immunity to noise normally found in the 030-MHz return path band, improved signal-to-noise ratio due to the processing gain, and utilization of the microprocessor normally found in the cable TV box to carry out the SS encoding operations.

As you saw in the previous patent, the ability to despread a received signal is dependent upon having a local copy (at the receiver] of the PN code used at the transmitter. In addition, this code generator must be perfectly synchronized to that of the transmitter. Much effort and complexity in SS designs relates to generating this synchronized PN code. Abstract 2 for a patent by AT&T Bell Laboratories proposes to solve this problem by eliminating the local code generator completely! Instead, they send the code itself along with the data. There are a number of means for accomplishing this feat. Different frequencies/

Patent Number	4,964,138
Issue Date	19901016
Inventor(s)	Nease, Greg A.; Cripps, Peter K.
State/Country	NJ
Assignee	Agilis Corporation
US References	3,337,803 3,864,635 3,917,999 4,112,372 4,214,209 4,291,409 4,291,410 4,387,465 4,423,517 4,545,061 4,561,089 4,567,588 4,649,549 4,653,069 4,672,658 4,760,586
Title	Differential correlator for spread spectrum communication system
Abstract	A spread spectrum communications system includes an encoder for differentially encoding a spread spectrum spreading code sequence in accordance with an input data signal. Each chip of the spreading code sequence is inverted, or not inverted, relative to the polarity of a corresponding chip of the spreading code sequence a fixed time delay previously, depending on whether the input data is a logic one or zero, respectively. At the receiver, the data is recovered in a differential data decoder wherein the presently received chip of the spread spectrum signal and a corresponding previously received chip of spread spectrum signal, received a fixed time delay previously, are compared one chip at a time. Since it is the spreading code sequence that is differentially encoded and differentially decoded, there is no need for a synchronized code sequence generator at the receiver, and data synchronization is achieved after one cycle time of the received spread spectrum spreading code sequence has elapsed.

PATENT TALK

bands for signal and PN code may be used, or the two components may be sent at two different times. Most likely, though, they would be sent in quadrature on the same carrier at the ***same*** time. Obviously, some of the secure communication capability of SS is compromised by doing this. For, if the code is known to all listeners, then anyone is able to despread the signal and recover the data. But the reduction in circuit complexity and associated cost, size, and power would make this unique approach of value in many applications where security is not the prime focus.

Abstract 3 addresses this same problem of creating a synchronized PN code source in a different way. Here, the Agilis Corp. proposes to actually transmit the PN code in a manner that it may be easily recovered at the receiver. As the code generating process proceeds, information is encoded on this PN code by inverting or not inverting chips in the code relative to chips offset by some fixed time. If initially all "0" data bits are sent, for example, then the PN code itself can be extracted at the receiver by simply performing an XOR function on a received chip with one that is offset by the same amount as in the encoder. Once the PN code is known, it can be used in the decoding process. The decoder simply has to (continually) XOR the input chip with one that arrived at the fixed offset earlier, and then invert or not invert the local PN code accordingly. The result is the original data!

Often, the prime reason for using SS is to make the signal secure from intercept. The patent described in Abstract 4 from Sandia Labs uses SS in an underwater beacon system. In this case, it is absolutely necessary for the listener to have an exact copy of the PN spreading code. For without it, the signal appears to be just so much ocean background noise! Note also how this system benefits from the many advantages of SS. Signals are enhanced by the "processing gain" which provides increased range and jamming resistance. Also, by using a different PN code for

Patent Number	4,951,263
Issue Date	1990 08 21
Inventor(s)	Shope, Steven M.
State/Country	NM
Assignee	Sandia Research Associates, Inc.
US References	3,900,823 3,992,692 4,081,784 4,109,100 4,847,817
Title	Spread spectrum underwater location beacon system
Abstract	An underwater location beacon emits a continuous wave signal which is phase-shift modulated by a pseudo-noise, spread spectrum code. This signal is detected with an exact replica of the transmitted code. To an unauthorized observer without the replica, the transponder's signal is indistinguishable from background ocean noise. The spread spectrum code allows extraction of the signal from high levels of ocean noise, providing an increased detection range, jamming resistance, covertness, and unique signals for each pinger. The outputs of the surface spread spectrum receiver are used with automated location algorithms. Several receivers at different surface positions provide the underwater coordinates of the pinger's location.

Patent Number	4,977,577
Issue Date	1990 12 11
Inventor(s)	Arthur, James D.; Sanderford, H., Jr.; Rouquette, Robert E.
State/Country	CA
Assignee	Axonn Corporation
US References	4,222,115 4,225,935 4,241,447 4,313,211 4,317,204 4,351,064 4,360,801 4,361,890 4,361,891 4,387,465 4,392,220 4,400,790 4,418,393 4,425,661 4,435,821 4,455,651 4,468,784 4,470,145 4,472,814 4,475,215 4,484,335 4,511,887 4,530,008 4,550,312 4,583,090 4,594,580 4,661,804 4,728,935 4,734,680
Title	Wireless alarm system
Abstract	A wireless alarm system using spread spectrum transmitters, fast frequency shift keying, spread spectrum receivers, and computer with a display. The spread spectrum transmitter includes an oscillator coupled to a microprocessor with chip code generation means, preamble register, address register, and data register. The spread spectrum receiver acquires synchronization of the spread spectrum signal using a microprocessor coupled to the quieting, signal strength, or baseband output of the receiver, with a two-step algorithm. The steps comprise achieving a coarse lock and a fine lock to the spread spectrum signal.

PATENT TALK

each "pinger" (underwater transmitter), all transmitters may operate in the same frequencies at the same time without interfering with one another. A few, smart surface receivers-using a lot of digital signal processing-can each extract all of the signals within range, and from them determine the location of all the "pingers."

Abstract 5 describes a wireless alarm system which uses SS techniques. The problem of synchronizing PN codes again appears, and a two-step procedure for achieving coarse and fine synchronization using a microprocessor is mentioned. The numerous references should be useful for anyone interested in using SS in such an application.

Lastly, I found the patent discussed in Abstract 6 to be a somewhat amusing and certainly novel (yet useful) application of SS. No longer when playing golf must you only *estimate* your distance from the pin. By using a number of low-power SS transmitters located at each pin, and a hand-held decoding receiver, the golfer can get an instant readout of his distance. This application, once again, makes use of the many benefits of SS that we've seen above: the ability to use low-power transmitters, the sharing of frequencies by many transmitters, and the ability to do ranging based on precisely timed signals from multiple sources. In essence, this is a mini-GPS system brought down to Earth!

Obviously, we've only touched on a few of the many application areas for spread spectrum technologies. And I've only begun to mention the voluminous theoretical and design considerations involved. But I hope this has provided an introduction to those of you who have no familiarity with this rapidly evolving field, and might serve as a source of reference material for further exploration.

Russ Reiss holds a Ph.D. in EE/CS and has been active in electronics for over 25 years as industry consultant, designer, college professor, entrepreneur, and company president. Using microprocessors since their inception, he has incorporated them into scores of custom devices and new products. He may be reached on the Circuit Cellar BBS or on CompuServe as 70054,1663.

SOURCE

Patent abstracts appearing in this column are from the Automated Patent Searching (APS) database from:

MicroPatent
25 Science Park
New Haven, CT 065 **11**
(203) 786-5500 or (800) 648-6787

422 Very Useful
423 Moderately Useful
424 Not Useful

6

Patent Number issue Date	5,056,106 1991 1008
Image Disc #	This patent is on PatentImages disc# 1991\079
Inventor(s) State/Country	Wang, James J.; Grayson, Robert M. CA
US References	3,150,372 3,868,692 4,136,394 4,297,701 4,480,310 4,665,404 4,698,781 4,703,444 4,731,613
US Class Int. Class	375/1 3421450 3421451 3421458 3421463 273/32B 273/32H 273/213 370/18 3641561 3641460 GO1 S 5/12 A63B 57/00
Title	Golf course ranging and direction-finding system using spread-spectrum radio location techniques
Abstract	The invention disclosed herein is directed to a method and apparatus which employs a spread-spectrum based radio location system, using handheld receiver units and fixed-position reference transmitters, to determine distance and direction between a golfer and key locations on a golf course, such as the distance and direction to a particular pin. The plurality of timing reference transmitters which are located throughout the vicinity of the golf course broadcast a spread-spectrum ranging signal consisting of a radio-frequency carrier directly modulated by a periodic pseudonoise (PN) coded or similar sequence. Each transmitter broadcasts at the same RF signal but a unique PN-coded sequence is assigned to each transmitter. Golfers are provided with the handheld receiving unit which receives the transmitter spread-spectrum signals and which synchronizes to the spread-spectrum signals in order to obtain range estimates to a selected set of reference transmitters. The handheld receivers also include memory to store the coordinates of the reference transmitters and the pin positions and other reference points for each hole on the golf course, which are either preloaded into memory or transmitted (as modulating data) with the ranging signal. Each handheld unit also includes a digital processor which incorporates a hyperbolic location algorithm to compute the handheld unit position based on the estimated ranges to the selected transmitters and the reference transmitter coordinates. The distance and direction from the current position to the pin or other selected reference points is then displayed via an appropriate medium on the handheld unit.

CONNECTIME conducted by Ken Davidson

The Circuit Cellar BBS
300/1200/2400/9600/14.4k bps
24 hours/7 days a week
(203) 871-I 988-Four incoming lines
Vernon, Connecticut

This month, we're going to ignore computers (to some extent) and concentrate on radio and chemistry. In the first thread, we get a lesson in radiated RF, not the kind that radiates from a computer, but the kind that can interfere with a computer.

Next, we follow up on the robotics threads from a few months ago and look at what if fakes to make an inexpensive radio modem to connect a robot with a base station.

Finally, as a follow up to the engine monitoring thread from last month, we discuss the actual engine chemistry and how if affects performance and emissions.

RFI affecting the computer

Msg#:15855
From: **ROBBIE LAIRD** To: ALL USERS

Has anyone had any experience with interference from HF radios affecting computers? The radio in question is 300 watts PEP operating on frequencies from 1.8 to 25 MHz. I know there can be interference in some cases. Has anyone ever heard of it causing permanent damage to a hard drive or a monitor? I would be especially interested in hard data, field strengths, and so forth.

Msg#:16132
From: MICHAEL MILLARD To: ROBBIE LAIRD

The type of radiation you are referring to here is known as nonionizing radiation, and, yes, it has been known to cause the sorts of trouble you are experiencing. It is difficult to offer advice on the matter because every case is different. In your case, I gather the offending transmission is a single-sideband transmission, in which case your calculated wattage may not actually reflect the peak-isotropic radiated power. But let's assume it's under a kilowatt in any event.

One of the first things we need to know is the proximity of the radiation source(s) to the devices being interfered with. If the two are located VERY close to each other (and presumably, you) you may also want to consider if the HF transmission represents a hazard to personnel and that the station does in fact comply with OST Bulletin #65 regarding human exposure limits to nonionizing radiation. (I have a spreadsheet that will make this determination easier if you know enough about the HF antenna system.) In which case, moving the HF "source" farther away to solve one problem may end of solving them both. (You can get OST-65 from the FCC's Office of Science & Technology or any government bookstore.)

But let's assume that the antenna is NOT "underfoot" and that you are under the guidelines of OST-65 with a field density of not more than $1\,mW/cm^2$ in the general vicinity of the malfunctioning equipment.. .

Normally, that power level would not cause objectionable or permanent impairment to most types of equipment. Shielding may help equipment that was poorly designed to work in moderate RF fields. (Another way of saying this is that crappy equipment works like crap. You may find that some other manufacturer's gear of the same type will work when others won't.) Part 15 of the FCC rules will give you a little more indication of what should be expected to work, and what should not be in the presence of the RF levels we are assuming here.

Now, the real meat and potatoes. Perhaps the root cause is "conducted" noise and is not in fact a direct radiation problem at all. You can check this by eliminating as many "outside paths" as practical to see if the interference subsides in any way. Running from a UPS, disconnecting and terminating any used or unused inputs and outputs, and trapping RF at the ports are all good checks. You may also have a strong localized field in the vicinity in which your equipment is located. This may be caused by ground reflections of the HF transmission (1.64 x the radiated field, typical) or by more local reflections caused by the building's construction or somesuch. Not too much you can do about this unless you have access to the HF antenna design/orientation. Remember that low frequencies such as these typically do not have high-gain antennas, and thus the vertical field component of the RF signal is very broad. The long wavelengths involved make higher-gain systems impractically large. So look for vertical fields (straight down the tower with no reflection to be around 60% or more). This number may be even higher if the HF antenna design radiates in both the vertical and horizontal planes (which I doubt it does.)

One last note, you might check with the HF operator to see if the antenna(s) system(s) is/are "loading-up" properly. A high VSWR can cause harmful reflections (not only to you, but to the RF gear). Correct the match to see if that helps. If the antenna doesn't load too well, a worthwhile

check might be to adjust the match to see if your problem gets any better or worse. Obviously, the HF transmitter itself ought to be in good working order.

But above all, KEEP IN MIND, that you may be dealing with a "coupling" problem which may change from day to day. Perhaps a dry atmosphere will offer less attenuation on one day than the morning after a strong storm.(?) It's important to look at real improvements over time to evaluate the effectiveness of any presumed "fix."

Inexpensive radio modems

Msg#:15271
From: DERRY BRYSON To: ALL USERS

A friend an I are building small (not tiny] mobile robots and are trying to work out a CHEAP radio modem setup. Our robots pack PC motherboards, floppies, and so forth, so we decided to try to use standard internal modems and cordless phones.

We were able to connect two modems (and computers) together using a 9-V battery spliced into a small piece of phone line [to sort of simulate a phone line). We then took this a step further and built another of these simulated phone lines and connected a phone to each modem and then laid the handsets next to each other with ends reversed. We were easily able to connect at 2400 bps (as long as no one whistled).

Anyway, the next step was to connect a modem to the base of a cordless phone. Now the problem: how do we wire the cordless phone handset into our simulated phone line for the other end?

We have tried using a transformer scavenged from a dead modem with one side hooked to our simulated phone line and the other hooked to the mic, speaker, and ground from the handset (i.e., the mic and speaker leads are connected together through pots to one lead on the transformer, ground is connected to the other lead on the transformer). Doesn't seem to work.

We have considered using a cheap phone and connecting the mic and speaker leads from the cordless phone handset to the speaker and mic leads on the cheap phone, respectively. Either directly or maybe through isolation transformers (Radio Shack S-ohm to 1000-ohm). Then, of course, connect the cheap phone to the modem on the robot.

Msg#:15285
From: PAUL PETERSEN To: DERRY BRYSON

Why not just use a cheap walkie-talkie from Radio Shack? They have a 46-MHz version for $10 or so. It's a kiddie model, but it does two-way communication with a low-power transmitter and receiver. There's even a button for sending Morse code beeps. Are you circuit design proficient? Have you tried light beams instead of radio waves? Sounds like a fun project.. .

Msg#:15358
From: DERRY BRYSON To: PAUL PETERSEN

I bought two walkie-talkies from Radio Shack (49 MHz) and have even built some boards that connect to a serial port with tone generators and decoders (567s) which almost work. These have three problems: the communication is only half duplex, the 567s are slow and I don't think I will be able to achieve anything beyond 300 bps, and there is interference on the band that they use in my area.

Communication using light has the problem that it won't travel through walls, which means putting transmitters in each room and running a lot of wire.

If we could just figure out how to couple the cordless handset to a phone line we would be set.

Msg#:15373
From: J. DEBERT To: DERRY BRYSON

I started a similar project some time ago (although I never finished it], so I may have a useful suggestion.

I chose the AM7910/7911 as the modem chip to get 1200 bps (Bell 202) and ran it through a buffer to a Radio Shack 49-MHz 2-way. I realized there would be problems from interference caused by other radios, so I chose the 5-channel hand-held. The hardware is connected to a microcontroller which controls all of the modem chip functions and transmit/receive and channel selection. I haven't worked out the details of how to establish a link and all, but I'm certain that it's possible-unless all the channels are occupied. In most areas, it's unlikely that all the channels would be in use close enough to you to cause bad interference. Since this design allowed only for simplex communications, owing to the use of only one channel, I thought of using some kind of "packet" style of communication, where the sender sends a packet, stops transmitting, then the receiver transmits a response, and so forth.

Msg#:15452
From: DERRY BRYSON To: J. DEBERT

These chips sound interesting, are they readily available? I might want to try them with the walkie-talkies I have when I build my next robot. I am planning on making it much smaller and basing it on an 805 1 which has serial ports, but would need the modem part.

As I mentioned in my previous message, I have built some circuits using 567s as tone decoders (and generators, as well), but the 567 is too slow (needs something like 10

wave fronts, which means maybe 300 bps using a 3000-Hz tone]. The modem chips sound like something to try, but I wonder about the bandwidth or distortion on the walkie-talkies. Can they faithfully transmit and receive the signals from these chips?

Msg#:15518
From: J. DEBERT To: DERRY BRYSON

The bandwidth of the radios should be more than sufficient. The modem chip-the AM7910—is a voice-band modem designed for telephone use. It is used also in amateur packet radio systems. I don't recall off-hand what all the modes are, but I remember that Bell 103, Bell 202, CCITT V.21 are available. It can be connected directly to your micro's bus with address decode circuitry.

Msg#:15556
From: MICHAEL MILLARD To: PAUL PETERSEN

A simple observation.. .

$$\frac{1}{bps} = \frac{1}{bandwidth} = \frac{1}{\$\$\$} \quad \frac{performance}{1} \quad \frac{0.0001}{aggrevation} \quad \frac{f(c)}{f(p)}$$

where

f(c) = chances of the damn thing working at all much less on the first try

f(p) = the number of people likely to notice

Msg#:15557
From: MICHAEL MILLARD To: DERRY BRYSON

If you're looking for something off-the-shelf, here's an interesting ad I ran across last week.. .

Not something I normally do, but let me quote verbatim out of an ad in last month's issue of *Wireless Design and Development,* page *52:*

"The Airlink Wireless Digital Modem model, the 64MP, makes it possible to create networks consisting of point-to-multipoint links over urban, suburban and rural areas, without a license. The 64MP products operate in one of the FCC-designated Part 15 (902-928 MHz) frequencies. They can be used quickly and painlessly to replace or extend multidrop wireline modems or licensed multiple address radio systems. The device offers synchronous and asynchronous operation at data rates up to 64 kbps. Cylink, 3 10 North Mary Ave., Sunnyvale, CA 94086"

Another option might be to look into an outfit called Monicor Electronics (305/979-1907). They make an FM (450-470 MHz) unit which comes with software for up to 48 nodes. Also supports X.30 for DEC and IBM hosts and LANs. Costs start around $500 for the two-board set. But that's serious hardware.

If you are sending simple commands (and I gather you are not or else you wouldn't want high data speeds), DTMF is always a simpler solution, at least when it comes to transmitter interfacing.

Msg#:15678
From: DERRY BRYSON To: MICHAEL MILLARD

Both of these sound too expensive. This is just a hobby robot after all.

Basically, I will be sending simple commands. Something like Right 90, Forward 100, Right 45, and so forth (actually, probably more like R90F100R45), and requesting simple information from the robot like a distance read from a sonar ping or maybe a whole 360" scan. It is too bad that DTMF has only 12 combinations, if there were 16 we could send a nybble at a time.

We have actually got it pretty much working now. I can only achieve 1200 bps error free with my cordless phone (my friend's phone will do 2400 bps), but I think that will be fast enough for now. We are now able to remotely pilot the robots using CTTY and running the programs we used before that required typing commands from the keyboard. Pretty exciting.

Msg#:15703
From: MATTHEW TAYLOR To: DERRY BRYSON

Sounds like you're having success currently, so this probably won't help, but DTMF DOES have 16 combinations. It is really a four-by-four matrix and I'd bet that most encoders / decoders support all 16 since all the HW is already there to do it!

Gasoline engine chemistry

Msg#:15105
From: GARY OLMSTEAD To: ALL USERS

On the topic of gas engines, what is the difference between "air-fuel ratio" and "lambda"? I have formulas for both and they are very similar. The formulas calculate the desired result by measuring the ratios of the various exhaust gases. The formula for air-fuel ratio is not reliable for oxygenated fuels; does this hold for lambda?

Msg#:15173
From: THOMAS BARNETT To: GARY OLMSTEAD

Lambda means the "excess-air ratio" and is given as lambda = quantity of air supplied / theoretical requirement = 1. A lean mixture (lambda > 1) contains more air while a rich mixture (lambda < 1) contains less air. Spark-ignition engines attain maximum power with an air deficiency of 0–10% (lambda = l-0.9) and minimum fuel consumption with roughly 10% excess air (lambda ≅ 1.1).

CONNECTIME

Msg#:15291
From: GARY OLMSTEAD To: THOMAS BARNETT

Thank you for the response. Just one more thing: Air-fuel ratio falls apart for oxygenated fuels; does this also hold true for lambda?

Msg#:15555
From: THOMAS **BARNETT** To: GARY OLMSTEAD

I'll try to explain the lambda ratio as best as I understand it as it applies to oxygenated fuels. First, the value of lambda between a rich mixture and a lean mixture will change only between 0.975 for a rich mixture and 1.025 for a lean mixture. This may vary plus or minus a few hundredths. That value is determined by the lambda sensor housed just ahead of the catalytic converter in the system exhaust. Part of the lambda sensor is housed within the exhaust system, the other part is exposed to ambient air.

As the sensor heats up to a temperature of approximately 300°C, it begins to conduct oxygen ions. There is a certain amount of residual oxygen expelled after combustion. If there is a difference between the electrodes of the sensor [i.e., the one exposed to the exhaust and the one exposed to ambient air), a voltage jump will occur indicating whether the mixture is richer or leaner than lambda = 1. [That, by the way, is called stoichiometric mixture if you're interested in big words.) This window of change is very small, occurring between 1 and 0.001.

That information is used by the vehicle computer to determine whether more or less air is required for combustion. Normal air-fuel mixture is determined by the throttle setting, so the lambda ratio will either add to or take away from the throttle setting. Enter oxygenated fuels.

The reason for adding oxygenates is to reduce carbon monoxide (CO) and hydrocarbons (HC). With oxygenated fuel, the engine excess air factor is lambda $\cong$ 0.9, so this will remain fairly constant because the oxygenates will keep the large excess air factor or change from occurring. Since engine operation is already in the rich range (lambda <1), you will get lousy acceleration and poor fuel consumption, which you've probably noticed. On the other hand, you've gobbled up most of the CO and HC. That's my best shot.

Msg#:15624
From: BRAD DAVIS To: THOMAS BARNETT

I know the message wasn't to me, but I was "lurking" and would like to thank you for the message. I experience about a 20-25 % decrease in fuel economy when using the oxygenated fuels in the winter. I live in Colorado Springs (about 6000+ feet) and the air is already thin enough that my acceleration is much less than what it is when near sea level. Anyway, here's the question: do the oxygenated fuels decrease CO and HC emissions by more than 20-25 % ? I would hope that since I need to burn an additional 20–25% of gas to go somewhere (at a reduced acceleration), the HC and CO emissions would drop by more that a few percent.

Msg#:15666
From: THOMAS BARNETT To: BRAD DAVIS

This is a tough question to answer because a number of things are happening. It also depends on how old the vehicle is and what type of catalytic converter is installed. Basically for most newer vehicles there is almost complete CO and NH_3 elimination by catalytic afterburning. Complete combustion would produce only carbon dioxide and water, both harmless byproducts, if you in fact had complete combustion. It is easy to see that that simply does not happen-just look at the sky.

There are many incomplete combustion byproducts—paraffins, olefins, aromatic hydrocarbons, aldehydes, ketones, carboxylic acids, acetylene, ethylene, hydrogen, soot, polycyclic hydrocarbons, oxides of nitrogen-and if you use leaded gasoline the added burden of lead oxides and lead halogenides along with fuel impurities such as sulfur oxides. Then insert a little sunlight and come up with organic peroxides, ozone, and peroxyacxetyl-nitrates. YOU probably need to be a chemist to know what all that means.

Here in Las Vegas, it means brownish-green haze that hangs over the city and only leaves when the wind blows at 40 MPH or more. So to answer your question, probably not. Pray for a high wind!

We invite you call the Circuit Cellar BBS and exchange messages *and files* with *other* Circuit Cellar *readers. It is available 24 hours a day and may be reached at (203) 871-1988. Set your modem for 8 data bits, 1 stop bit, no parity, and 300, 1200, 2400, 9600, or 14.4k bps.*

ARTICLE SOFTWARE

Software for the articles in this and past issues of *The Computer Applications Journal* may be downloaded from the Circuit Cellar BBS free of charge. For those unable to download files, the software is also available on one 360K IBM PC-format disk for only $12.

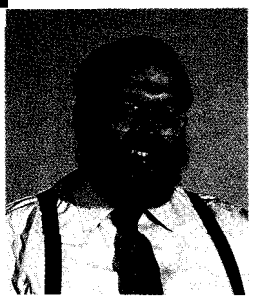
To order Software on Disk, send check or money order to: The Computer Applications Journal, Software On Disk, P.O. Box 772, Vernon, CT 06066, or use your VISA or Mastercard and call (203) 8752199. Be sure to specify the issue number of each disk you order. Please add $3 for shipping outside the U.S.

I R S

425 Very Useful 426 Moderately Useful 427 Not Useful

STEVE'S OWN INK

The Collegiate Challenge



I can hardly believe it but September has rolled around already. Another summer has passed and fall is about to burst onto the scene. September always reminds me of back to school. Could it be all of those ads I see that make me think that way? Maybe.

To all of you who are going to back to school, I hope your professors took advantage of the special deal we offered to colleges and universities. If so, I hope you enjoy reading the *Computer Applications Journal* along with all of the materials you will be expected to read this semester. I also hope you have a successful semester.

Speaking of successful semesters, I can't emphasize enough how important it is that you apply yourself. Let me tell you a tale of two students. One student stayed out late every night playing around, doing just enough to get by with slightly better than average grades, Content with his lot, this student couldn't wait for all of this agonizing homework to end so he would never have to think about those complicated subjects ever again. Another student faces the same material as a personal challenge, trying to learn all he can; trying to absorb as much as he can; trying to rise to the best of his abilities.

Well, finally the day comes that it is over. Graduation. I won't give you the song and dance about which one of these two impressed the job interviewers and received the better offer. That would be trite. Instead, I'll ask you to consider which of these students has formed better self-development skills, Which of these two do we want the country and economy to depend on?

I won't tell you which of these students I was. Let's just say I saw the light very early and decided a career as a bum was self-defeating. Instead, I ask you to examine yourself to see which kind of student you are. Take this to heart, because after you are out of the collegiate program, the rest of your intellectual development is up to you. Perhaps you will get "lucky(?)" and land a job in an organization where all of your efforts are strictly guided by the needs of the company. Perhaps you'll get "lucky(?)" and land a job in an organization where you are in charge of whatever you do. Of course, the responsibilities in the latter job mean you get the "bullet" as well as the praise.

I always prefer the second career path because that's just the kind of person I am. Personally, I couldn't begin to think of myself in any kind of position where I couldn't apply unrestricted drive and motivation.

So, try to see beyond the immediate tasks of homework or dull career assignments. Look for a deeper lesson in this challenge you have accepted for yourself. Is your challenge, "How little can I do and still manage to sneak past?" or is your challenge, "How good at this can I become?" Be true to yourself and honestly appraise your approach.

Those of us who don't have to go through the grueling rigors of classwork (thank the stars) sympathize with you, but the end result puts you in a minority among men (and women). How long you stay there just depends on how you apply it.