CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

#56 MARCH 1995

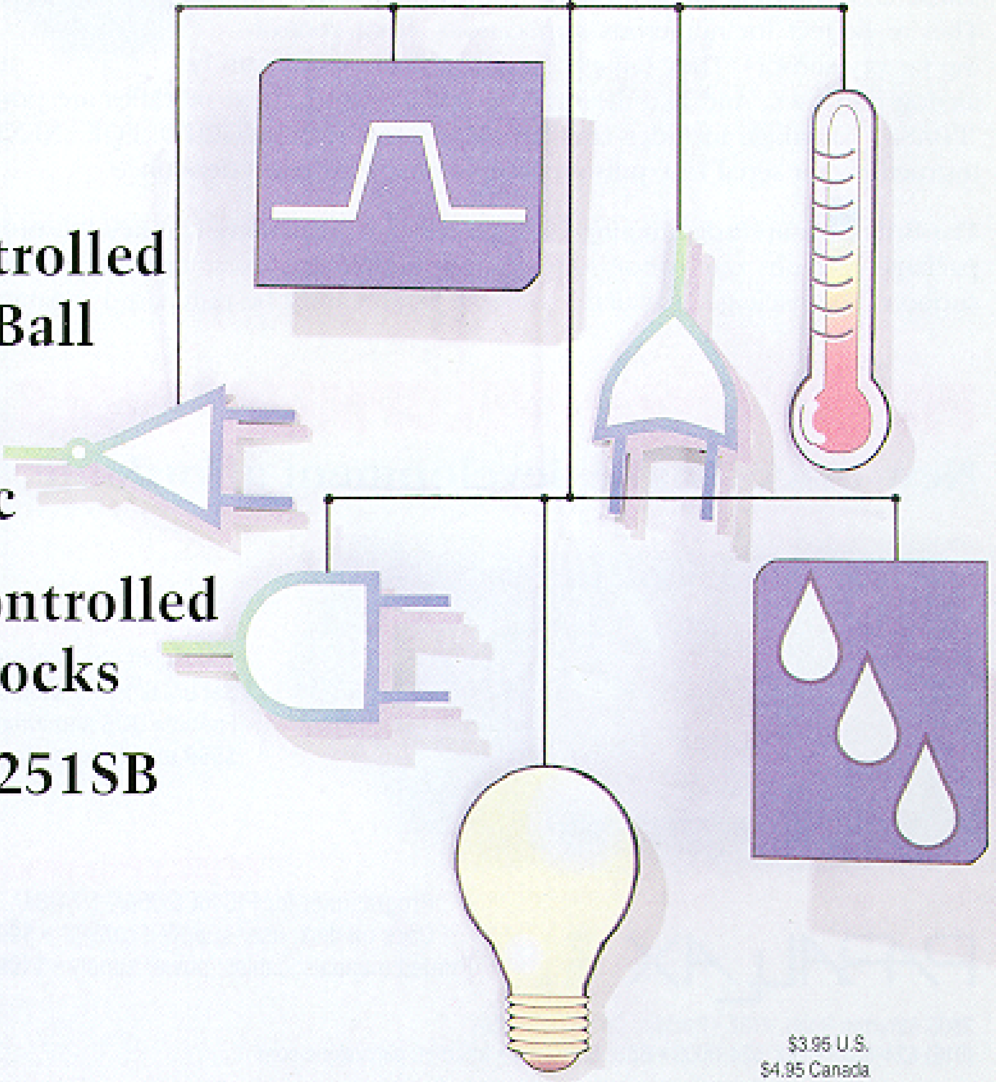
FUZZY LOGIC

Fuzzy-Controlled Levitating Ball

Embedded Fuzzy Logic

Remote-Controlled Car Door Locks

Intel's 8xC251SB



03

0 74470 75349 0

$3.95 U.S.
$4.95 Canada

EDITOR'S INK

Warm Fuzzies?

fuzzy logic is an idea whose time has come. The` Japanese and Europeans are far ahead of U.S. designers in embracing this novel programming approach. It's about time more engineers started to take notice.

When I sat down to choose themes for 1995, I had one fuzzy logic article in hand, so decided to gamble and devote an issue to the topic. I'd been seeing it discussed elsewhere on a more regular basis, so thought it was time for us to take a look. I was pleasantly surprised when I ended up with three substantial articles and an extensive sidebar describing fuzzy logic and its applications, We even have a fuzzy-based home controller article waiting in the wings for a future issue of Home *Automation &* Building Control.

I guess I shouldn't have been surprised. The methodology more closely approximates the way humans make decisions, so resulting control systems behave more "naturally." As more people become aware of its advantages, more writers will talk about it. I learned a lot working on this issue, and I hope you do, too.

Our first fuzzy article presents a good overview of what the technique is and how it can be applied to a simple temperature feedback system. Can fuzzy logic truly be called "the next generation of control"?

Next, we narrow the focus to see how fuzzy logic can be simplified enough to be effective on a small microcontroller with limited resources. It doesn't necessarily require lots of horsepower as was thought in the early days.

Our next article continues on the seemingly never-ending quest for the best way to levitate a ball on an air cushion. Tom Cantrell originally threw down the gauntlet and presented one approach based on PID algorithms. However, fuzzy turns out to be just as effective and requires less tweaking. The sidebar to this article presents yet another approach to the same problem.

Our final feature is a follow-up to another past article. In the February '94 issue, Do-While Jones cautioned against the indiscriminate use of interrupts. Mike Smith shows that under the right circumstances, software interrupts can be quite effective.

In our columns, Ed continues enhancing his multitasking kernel by providing a way for low-level tasks to access high-level routines. Jeff comes to Bev's rescue with an RF-based link to her car that enhances her safety in dark, lonely parking lots. Tom explores yet another 8051 derivative, but this one comes from the designers of the original 8051—Intel. Finally, John explains how to put together a low cost but highly effective development system for the Atmel AT89C2051.

[illegible]

CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

FOUNDER/EDITORIAL DIRECTOR
Steve Ciarcia

EDITOR-IN-CHIEF
Ken Davidson

TECHNICAL EDITOR
Janice Marinelli

ENGINEERING STAFF
Jeff Bachiochi & Ed Nisley

WEST COAST EDITOR
Tom Cantrell

CONTRIBUTING EDITOR
John Dybowski

NEW PRODUCTS EDITOR
Harv Weiner

ART DIRECTOR
Lisa Ferry

PRODUCTION STAFF
John Gorsky
James Soussounis

CONTRIBUTORS:
Jon Elson
Tim McDonough
Frank Kuechmann
Pellervo Kaskinen

PUBLISHER
Daniel Rodrigues

PUBLISHER'S ASSISTANT
Sue Hodge

CIRCULATION COORDINATOR
Mansella

CIRCULATION ASSISTANT
Barbara Maleski

CIRCULATION CONSULTANT
Gregory Spitzfaden

BUSINESS MANAGER
Jeannette Walters

ADVERTISING COORDINATOR
Dan Gorsky

CIRCUIT CELLAR INK, THE COMPUTER APPLICATIONS JOURNAL (ISSN 0896-8985) is published monthly by Circuit Cellar Incorporated. 4 Park Street, Suite 20, Vernon, CT 06066 (203) 875-2751 Second class postage paid at Vernon, CT and additional offices. One-year (12 issues) subscription rate U S A and possessions $21 95, Canada/Mexico $31.95, all other countries $49.95. All subscription orders payable in U.S funds only. via international postal money order or check drawn on U S bank. Direct subscription orders and subscription related questions10 Circuit Cellar INK Subscriptions, P.O. Box 698, Holmes, PA 19043-9613 or call (600) 269.6301

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

Cover photography by Barbara Swenson
PRINTED IN THE UNITED STATES

HAJAR ASSOCIATES NATIONAL ADVERTISING REPRESENTATIVES

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

SOUTHEAST
Christa Collins
(305) 966-3939
Fax: (305) 985-8457

MIDWEST
Nanette Traetow
(708) 789-3080
Fax: (708) 789-3082

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

Circuit Cellar BBS—24 Hrs 300112001240019600114 4k bps, 8 bits, no parity, 1 stop bit, (203) 871-1988; 24001 9600 bps Courier HST, (203) 871-0549

INSIDE ISSUE 56

NEW PRODUCT NEWS

Edited by Harv Weiner

MULTIPURPOSE EMBEDDED CONTROLLER

Remote Processing has released a multipurpose embedded controller with an operating system, operator interface, and I/O on a single card. The RPC-320 uses the Dallas Semiconductor 80C320 CPU with a unique addressing scheme to access 1 MB of memory. Normally, the 80C320 (a variation of the Intel 8032) can only access 64 KB of RAM.

The RPC-320 features industrial I/O. It has 8 ADC input lines with 12-bit resolution, 20-MHz quadrature encoder and counter input, 34 digital lines, and 2 RS-232 or RS-485 serial ports. A keypad and LCD character and graphics port for operator interface are also included. Over 12,000 lines of code can be stored and executed to a flash EPROM by the oncard RPBASIC.

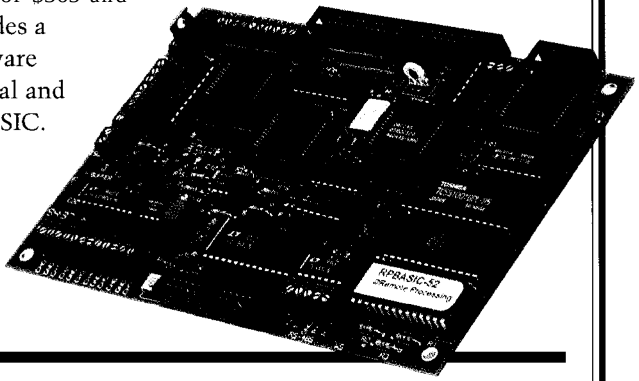
RPBASIC is an improved version of Intel BASIC-52 which directly supports the hardware using single commands. The included RPBASIC operating system accesses up to 1 MB of RAM and flash memory for data logging and program storage. Over 500 KB of programs can be autorun on powerup.

A built-in temperature transducer monitors ambient temperatures. Two operational amplifiers buffer, amplify, and filter inputs from sensors. The temperature transducer and amplifier can be connected directly to an ADC input, which accommodates eight single-ended or four differential inputs with ranges of O-5 or f2.5 V. Inputs are overload protected to ±25 V.

The 20-MHz multimode counter interfaces to quadrature encoders or other high-frequency devices. Up and down counters interrupt the program when a preset count is reached. RPBASIC loads or reads a 24-bit number to the counter. Frequency measurements are possible with RPBASIC's multitasking feature.

Many LCD displays interface to the display port. RPBASIC positions the cursor and writes to the display in a single command. Graphics commands draw lines and control pixels to show level or position. BASIC also scans and buffers entries from a 16-key keypad port.

The RPC-320 sells for $365 and includes a hardware manual and RPBASIC.



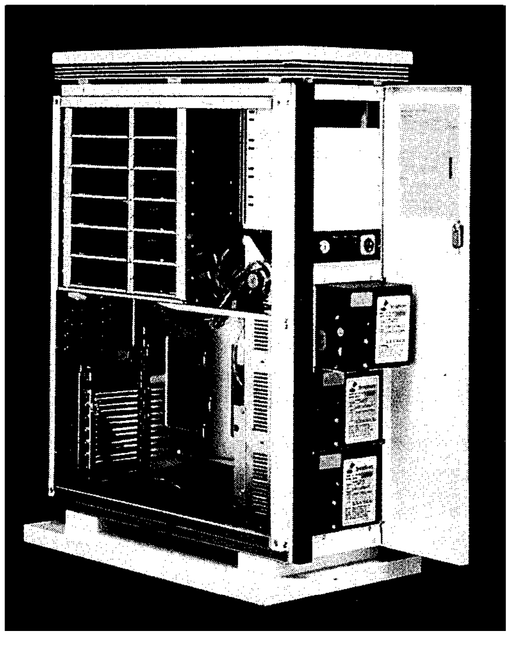
Remote Processing
6510 W. 91st Ave.
Westminister, CO 80030
(303) 690-l 588
Fax: (303) 690-l 875

#500

FAULT-TOLERANT POWER SUPPLY

A line of active fault-tolerant power supplies which provide mission-critical PC users with automatic backup power has been introduced by Antec. Available as a tower enclosure (**KS022**), file server tower (**KS033**), and disk array tower (**KS044**), the Reliant products are the first power supplies which cause absolutely no power interference to computer components.

Antec is also introducing the RPT-600, a unique AT-size, fault-tolerant power supply designed with two built-in, 300-W power units. The RPT600 works as a highly efficient 600-W supply or as a loadsharing, fault-tolerant unit with two individual 300-W supplies.



The Reliant immediately takes over when a master power supply fails to prevent damage due to overvoltage, overcurrent, and other power problems. If a master power supply fails, sophisticated diagnostics sound an alarm, alerting the user to hot-swap the failed power supply.

The RPT-600 sells for $199, the KS022 for $799, KS033 for $1699, and the KS044 for $1799.

Antec, Inc.
2859 Bayview Dr. • Fremont, CA 94538
(510) 770-1200 • Fax: (510) 770-1288 **#501**

NEW PRODUCT NEWS

DATA ACQUISITION AND CONTROL SYSTEM

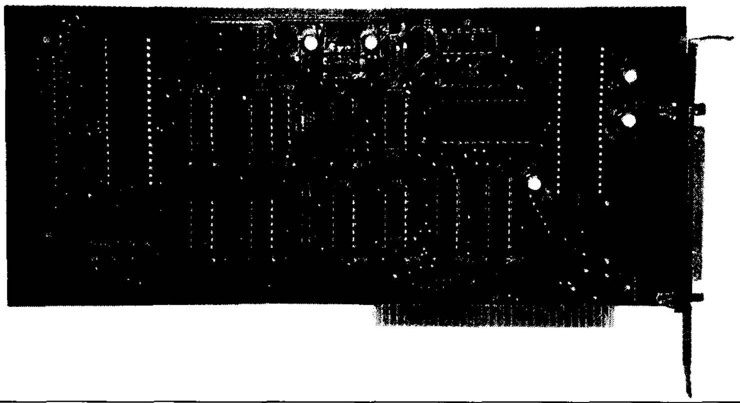
Prairie Digital has introduced a low-cost data-acquisition and control system for all ISA bus computers. The Model 100 provides the most commonly used features of analog and digital I/O boards. Typical measurements include temperature, pressure, humidity, light levels, force, and acceleration.

Through software, users can select eight single-ended channels or four differential channels. Conversions are performed in 10 µs with 12 bits of resolution. Users set an input range of O-5 V or ±2.5 V. Four channels of 8-bit analog output (O-5 V) can also be selected.

Twenty-four lines of digital input and/or output are provided (eight lines are semidedicated) for controlling relays, lights, motors, switches, thermostats, and liquid levels. Also provided are three 16-bit timer/counters for timing events, counting pulses, and generating interrupts with accurate timing.

Atlantis software enables the Model 100 to emulate strip-chart recorders, oscilloscopes, and digital voltmeters simultaneously. Up to ten instruments can be displayed at one time, including bar graphs and a real-time clock. Foreground or background sampling, software triggers, and user-definable macros are also featured.

Model 100 sells for $279. Optional Atlantis software sells for $79.



Prairie Digital, Inc.
846 17th St.
Prairie du Sac, WI
53578
(608) 643-8599
Fax: (608) 643-6754

#502

If, like Doc, you think bugs belong six feet under, then step up to Paradigm DEBUG and get the right weapon for the toughest '186 or V-series embedded application. Take 'em on by yourself with Paradigm **DEBUG/RT** or gang up on 'em with a popular in-circuit emulator. 'Cuz no matter what kind of Borland or Microsoft C/C++ vermin you're fightin', ya better not go in empty-handed or firing blanks from inferior weapons. Ya just might not live to regret it.

'Nuff said.

PARADIGM

Proven *Solutions* for Embedded *C/C++* Developers

1-800-537-5043

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

#103

NEW PRODUCT NEWS

UNIVERSAL FRONT-PANEL CONTROLLER

The **IQC816 Universal** Front-Panel **Controller** chip from IQ Systems is the first in a series of chips which greatly simplifies incorporating encoders, displays, keyboards, switches, sound, and speech in electronic products. The product family is founded on a new command set which provides a standardized user interface regardless of the host processor type.

IQC8 16 can support 32 digits of LED display addressed as 4 displays of 8 digits. Displays can be concatenated for more than 8 digits or reduced in width for increased brightness. An intelligent display offers features such as left and right formatting, choice of cursors, alpha decode, programmable display width, and so on. Thirty-two segments of LCD display plus 4 x 40 lines of alphanumeric smart LCD display are also supported.

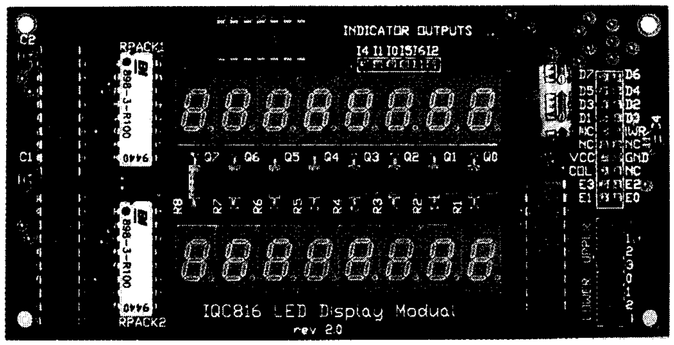
Sixteen nonmultiplexed outputs drive actuators, relays, or incandescent bulbs. Eight rotary encoders with full quadrature decoding are supported, and each encoder is supported by an 8-bit up/down counter. Forty-eight standard switches (176 with shift, control, and alternate key modifiers) can be supported. The output buffer is 8 bytes deep and records switch scan codes and the address of encoders which have changed state. A sound generator with programmable frequency and duration and an UART are also on the chip.

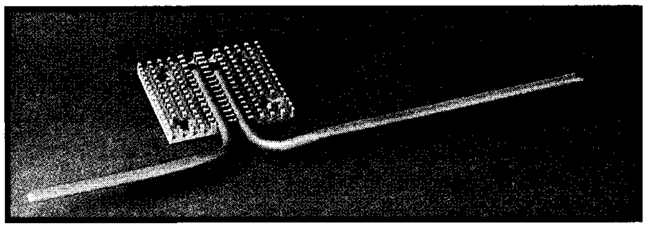
The software interface is flexible and easy to use. The goal is to standardize the user interface command set in much the same way as the interface to modems was standardized with the Hayes AT command set.

The IQC8 16 is available in a 40-pin DIP package and sells for $19.95 in single quantity and $9.95 in 1000s. An evaluation kit is available for $95. The kit, which contains C drivers for the IBM PC and numerous application examples, enables a designer to plug on displays, encoders, speakers, and so on, then couple them to any host with a UART.

IQ Systems, Inc.
20 Church Hill Rd.
Newtown, CT 06470
(203) 270-8667
Fax: (203) **270-9064**
Internet: **76636.3267@compuserve.com**

#503





PENTIUM COOLING SOLUTION

Thermacore Inc. has announced a new series of standard and customizable products designed to cool Pentium or other high-performance processors in notebook computers. The Processor Thermal Management Systems offer high-efficiency cooling without noise, moving parts, or electrical power requirements. The performance of these compact solutions helps chips such as the Pentium to maintain case temperatures at less than 40°C over the ambient temperature, thereby ensuring the chip's intended life and performance characteristics.

This new line of products uses Thermacore's exclusive heat-pipe technology. Through this passive technology, heat is moved from the chip's surface via a two-phase heat-transfer process which requires no moving parts or power from a notebook's battery. Depending on design and space considerations, heat can then be dissipated by natural convection using fins attached to the heat pipe or through existing components such as the keyboard's aluminum back plane, which can act as a heat sink.

Unlike other heat-transfer solutions, Thermacore's products have been designed from the ground up for the notebook computer. For example, they operate in any position the user places the notebook, including upside down. Its light-weight (less than 65 g for the full heat sink) and snap-on chip design optimize manufacturing while its mean time to failure is over 100,00 hours.

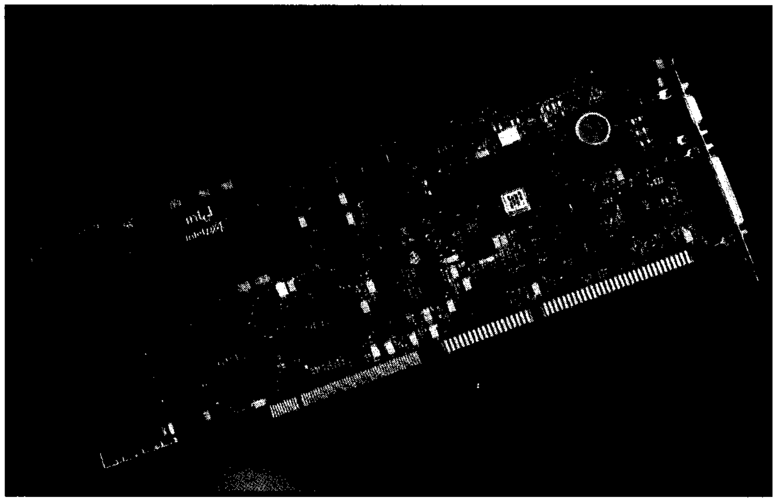
Thermacore's new products are available immediately. Designs can be modified to fit any style case or board configuration.

Thermacore, Inc.
780 Eden Rd.
Lancaster, PA 17601
(717) 569-6551
Fax: (717) 569-4797

#504

NEW PRODUCT NEWS

PASSIVE BACKPLANE SBC



The PCI-930 from Teknor Microsystems is a '486DX-based, passive backplane CPU card, operating at processing speeds of up to 100 MHz with Intel's DX4 microprocessor chip. The ICI-930 is designed to the new PICMG Rev 2.0 industrial PCI specification and offers full PCI and ISA passive backplane compatibility. This standard enables compatible cards to run on a shared PCI and ISA passive backplane.

Full onboard I/O, such as hard and floppy disk controllers, serial, parallel, and keyboard ports, and bootable flash memory, is standard on the card. SVGA resolutions to 1024 x 768 x 256 colors are included. OEM designers gain fast video access through a 32-bit PCI local-bus interface to Cirrus Logic's CL-GD5430 video-processor chip. High PCI transfer rates (up to 132 MBps) provide maximum throughput and help alleviate system bottlenecks. The card also includes a 256-KB write-back cache, a real-time clock with battery backup, AM1 BIOS, watchdog timer, SVGA output, and mouse port.

Configurable with up to 128-MB DRAM and up to 4-MB flash memory, the ICI-930 is well-suited for high-speed, performance-dependent applications such as medical imaging, telecommunications, and industrial automation.

PCI-930 microprocessor configuration options include an Intel '486DX @50 MHz, '486DX2@66 MHz, and '486DX4@100 MHz. The '486DX-50 entry-level version is priced at $1595.

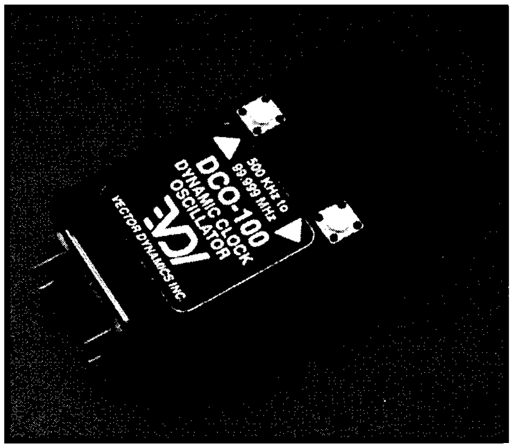
Teknor Microsystems, Inc.
616 **Curé** Boivin • Boisbriand, Quebec • Canada **J7G 2A7**
(514) 437-5682 • Fax: (514) 437-8053

#505

DYNAMIC CLOCK OSCILLATOR

Vector Dynamics has introduced the **DCO-100 Dynamic Clock Oscillator,** a self-contained test instrument for development engineers. This product emulates an oscillator toolbox, eliminating the need to stock multiple oscillator frequencies to meet the demands of new product design.

The DCO-100 provides user-selectable clock frequencies from 500 kHz to 99.999 MHz in 1-kHz steps. Changes in frequency are glitch-free, allowing on-the-fly frequency selection without removing it from the circuit under test. Frequency is selected by using two push buttons and is displayed on a 5-digit decimal display located at the top of the module. The module can be easily reprogrammed to any frequency within its range, making it a versatile development tool which can be reused on other projects.



The DCO-100 module has an identical footprint to a standard 4-pin (14-pin configuration) full-size, fixed-frequency oscillator and is powered by the 5 V normally provided to pin 14 of the clock oscillator in the circuit. The unit's dimensions are 1.27" x 2.75" x 0.5".

The DCO-100 includes a built-in display and programming buttons, so no accessories are needed. A user handbook with useful application notes and storage box are also included. The DCO-100 sells for $200.

Vector Dynamics, Inc.
1880 Tanglewood Dr. NE • **St.** Petersburg, FL 33702
(813) 526-7038 • Fax: (813) 527-6534

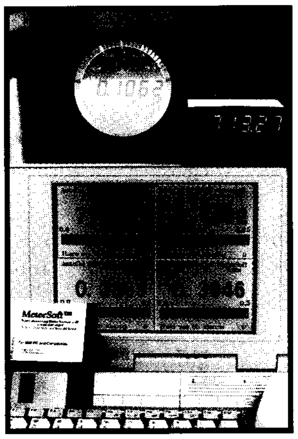
#506

NEW PRODUCT NEWS

MULTIPLE METER SOFTWARE

AGX has introduced a new, multiple-meter software product with drivers for most serial interface meters. Metersoft offers real-time display, meter monitor and logging, controllers, counters, bar graphs, sensors, and transmitters with an RS-232 or RS-485 serial interface. Metersoft displays 1, 4, 9, or 16 meters per PC screen with alarms, channel ID, engineering units, and trending information.

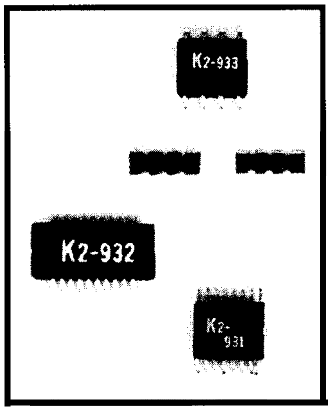
Metersoft provides such as uniform meter setup and configuration, data logging, multiple-meter display, test and measurement, quality control, and the creation of virtual meters (for example, you could determine the sum or difference of two real meters).



Priced from $99, Metersoft provides the user with a low-cost data acquisition and monitoring system for existing and newly purchased meters.

AGX Corp.
Metersoft Division
5761 **Uplander** Way
Culver City, CA 90230
(310) 642-6663
Fax: (310) 642-6661 **#507**

DATAACCESS ARRANGEMENT



The first all-silicon, full-function Data Access Arrangement (DAA) has been developed by Krypton Isolation. The K^2 DAA chip set provides isolation from 1.5 and 2 kVDC (3.5 kVDC for European standards) in stand-alone, card-level, and PCMCIA fax modems for computers ranging in size from portable and hand-held systems to desktop workstations.

As a total solution to the problem of isolating modem chip sets from the telephone line, the devices are easy to use, lower in cost than other forms of DAA, and function at all speeds up to V.32, V.32bis, and V.34. It is designed to mount inside any kind of fax-modem card housing without any modifications to the motherboard. The chip set connects directly to the telephone line.

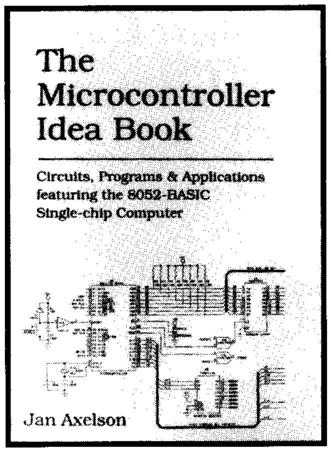
Other features of the K^2 chip set include off-hook relay control, ring-indication control, internal 2-to-4 wire conversion, and caller ID. K^2 offers a power-down mode, operation from a single 5-v source, and low power consumption. The chip set includes one device in an 8-pin SOIC package, one in a 16-pin QSOP package, and one in a 20-pin QSOP package. The chip set meets all appropriate standards.

The K^2 Chip Set sells for $8.50 in quantity.

Krypton Isolation, Inc.
39111 Paseo Padre Parkway, **Ste.** 202
Fremont, CA 94538
(510) 713-9100
Fax: (510) 713-9188 **#508**

MICROCONTROLLER BOOK

Lakeview Research has announced a hands-on guide of circuits, programs, and applications featuring the 8052-BASIC single-chip computer. Jan Axelson's ***The Microcontroller Idea*** Book presents practical designs for use in data loggers, controllers, and other embedded computer applications. In addition to the basic circuits needed for any project, the book shows how to add keypads, switches, relays, displays, sensors, clock/calendars, motor controls, wireless links, and other I/O interfaces.



The Microcontroller Idea Book
Circuits, Programs & Applications featuring the 8052-BASIC Single-chip Computer
Jan Axelson

This 277-page book includes complete circuit schematics, parts lists, design theory, construction and debugging tips, and program listings. Circuit and program examples are based on the popular 8052-BASIC microcontroller, whose on-chip BASIC interpreter includes over 100 commands, statements, and operators for convenient writing, running, and debugging of programs. Special commands store user programs in EPROM or other nonvolatile memory. As a member of the 8051 microcontroller family, the chip uses a standard, popular architecture.

The Microcontroller Idea Book sells for $31.95 plus $3.00 shipping.

Lakeview Research
2209 Winnebago St.
Madison, WI 53704
(608) 241-5824 **#509**

FEATURES

Fuzzy Logic: The Next Generation of Control

FEATURE ARTICLE

Bud Moss

As Bud points out, fuzzy logic balances many factors in measurable and mathematically precise ways. A myriad of graphs help visualize the fuzzy-logic decision-making process.

fuzzy logic is another step toward eliminating the need to reduce our thoughts to the point they become unrecognizable. It lets us design the core software of a control system or similar task using a method which simulates how we think. It's no longer necessary to break down our ideas into line after line of code or long, complex equations. Instead, we can use graphs and shapes, and a few IF.. .THEN rules complete the system.

Fuzzy logic is not for everyone—just most of us. If you enjoy generating and optimizing equations more than implementing and debugging the system, fuzzy logic may not be for you.

For the rest of us, it's well worth the effort.

ABSOLUTESVERSUS GENERALIZATION

Have you ever been hiking and had to cross a stream, but there wasn't a bridge?

My first reaction is to look for another way. Scanning the water, I search for rocks that can be used as stepping stones. To pick the right rocks, I jump into the water and document every characteristic of every rock using the tools-a protractor and ruler-I just happened to pack. Next, I generate a formula, plug the data into a calculator, and hope. Two or three days later, the stream is conquered and I'm on my way.

Not very likely.

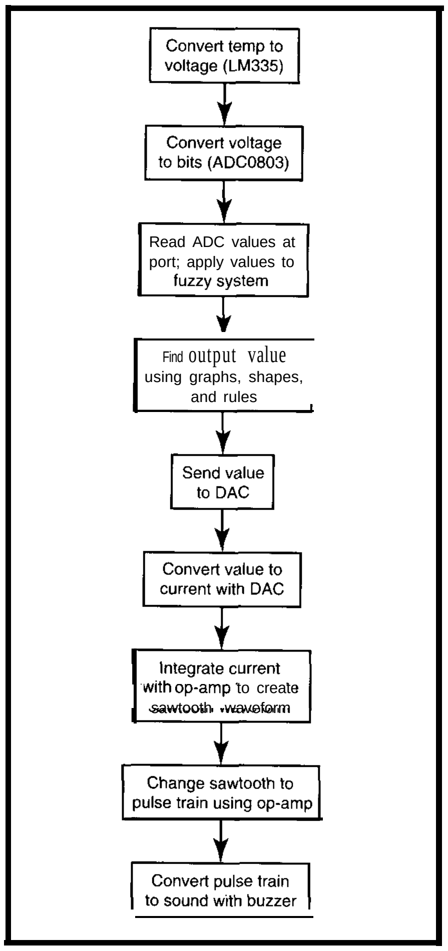
Instead, I base my decision on approximations. First, I decide which characteristics should be considered and how important they are compared to the others. I know that rocks with a large flat top are easier to stand on and the closer together they are the better. I also know that the distance between them is more important than the shape of the top. After finding how well each rock fits the categories, I choose the right one.

Fuzzy logic follows a similar process. It allows us to be imprecise and still arrive at the correct answer. By setting up graphs (size, top, and distance), adding shapes (large, flat, and close), and specifying the rules (IF the rock is small AND pointed OR is too far away, THEN eliminate it), I come up with the best rocks.

Unlike a system that uses absolutes, this system, once it is proven, enables me to find the correct rock in any stream, not just this one.

ITS BEGINNINGS

In 1965, Lotfi Zadeh found that, due to the contradictory nature of control systems, they were not easily represented using the traditional method of mathematical modeling. He theorized that adding imprecision to the system would allow it to react more precisely when presented with conflicting input data. His article "Fuzzy Sets" launched the field of fuzzy logic.



Convert temp to voltage (LM335)
Convert voltage to bits (ADC0803)
Read ADC values at port; apply values to fuzzy system
Find output value using graphs, shapes, and rules
Send value to DAC
Convert value to current with DAC
Integrate current with op-amp to create sawtooth waveform
Change sawtooth to pulse train using op-amp
Convert pulse train to sound with buzzer

Figure 1—The sample fuzzy logic system described starts with temperature and ends with sound. However, it doesn't simply pass the ADC output value to the input of a DAC.

Western countries didn't exactly jump when fuzzy logic was born. In fact, it took almost 30 years for it to hit mainstream media. There's probably a multitude of reasons, but most are related to the fact that the system could not be proven mathematically. Some believed that an unprovable system was the same as no system. Others were uncomfortable with the word fuzzy—an excellent word for what it describes, but to those accustomed to precision, it may not have been the best choice.

Unlike the West, the Japanese were interested. The word fuzzy was not translated, but transferred phonetically, thereby facilitating the acceptance of the concept at face value. With an easy acceptance, the Japanese went straight to the next step—experimentation, which quickly demonstrated that traditional methods of proof were unnecessary. For years, the Japanese have marketed fuzzy-based products.

But, the West is catching up. In the last few years, many companies have been developing products based on fuzzy logic concepts. Most large IC manufactures are marketing microcontrollers and coprocessors optimized to perform in a fuzzy-logic environment.

For example, did you know that fuzzy logic controls the Saturn automobile's transmission? The designers found that they achieved better performance and smoother operation with fuzzy logic than they did using a traditional approach. This trend should continue as more designers see the benefits of this type of control.

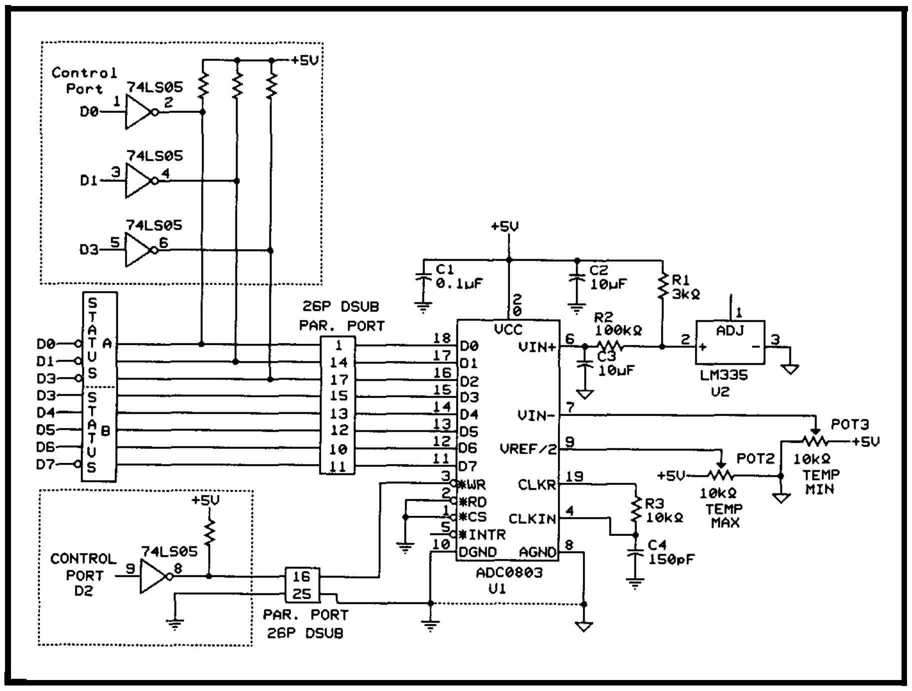


Figure 2—An LM335 converts temperature to a voltage which is digitized by the ADC0803. The digital value is transferred to the PC via one of the parallel ports.

THE HARDWARE

To gain an understanding of fuzzy logic, we'll use the simplest control problem: one input and one output. Figure 1 shows the flow of an entire system. We'll vary the frequency of a speaker as the room temperature varies.

The hardware was designed to show the ease of setting up a control system based on fuzzy logic. Most of

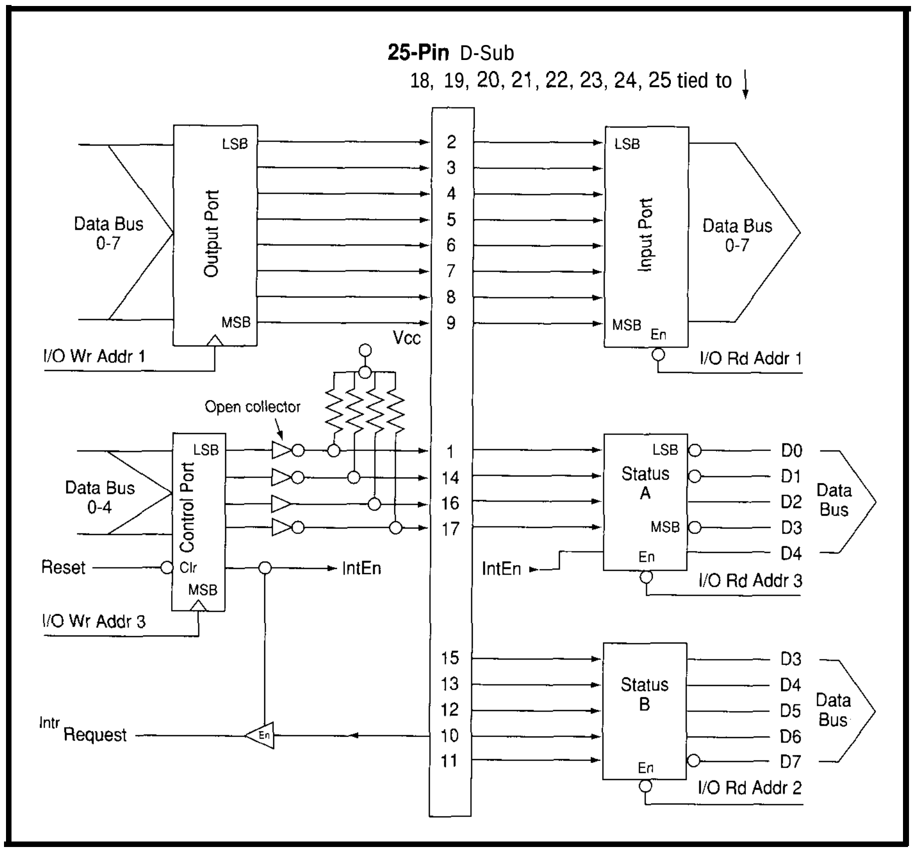


Figure 3—*Instead of using a board, the interface to ADC and DAC is done through a 25-pin D-sub printer* port *connector.*

us have an unused parallel port or an A/B box. Instead of using another slot in the PC, the hardware interface will be done via the PC's printer port.

TEMPERATURE TO VOLTAGE

The LM335 converts the room temperature into a voltage that's used to feed the ADC. Basically, it's a zener diode whose breakdown voltage varies proportionally with temperature. For each Kelvin degree, the junction voltage changes 10 mV. (Remember Kelvin begins at absolute zero, which is –459.67°F. To convert Kelvin to Fahrenheit, use the following formula: F° = 1.8 x Kelvin – 459.67°.) To hold the input stable, R2 and C2 are configured as a low-pass filter with a one-second time constant.

8-
analog-
640-kHz

$$\text{Frequency} = \frac{1}{1.1 \times R3 \times R4}$$

with R3 ≈ 10 kΩ. The range of input voltages is set using the V_{IN-} and $V_{REF/2}$ pins and the potentiometers connected to them.

For example, if the minimum input voltage is 0.5 V and the maximum is 3.5 V, the reference voltage is (3.5 – 0.5)/2

V_{IN+}1 with both

V_{IN-} at 0.5 V V_{IN+}1 d

1

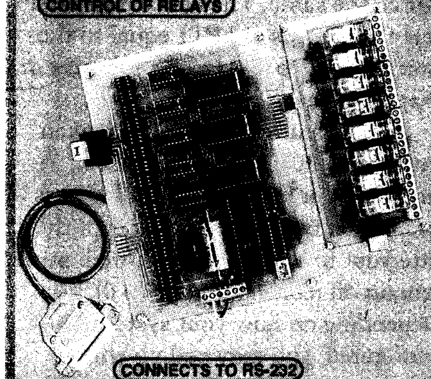
of the printer port's hardware. Table

Addr	Port 1	Port 2	Port 3
1	3BC	378	278
2	3BD	379	279
3	3BE	37A	27A

Table I--The *actual I/O port addresses used in your code depend on which physical parallel port you're* using. The Addr numbers correspond to the Addr labels *in Figure* 3.

RELAY INTERFACE

PROVIDES SOFTWARE CONTROL OF RELAYS

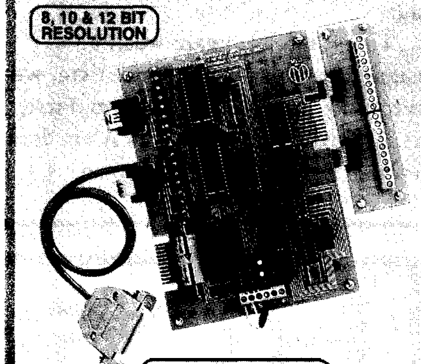


CONNECTS TO RS-232

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

ANALOG TO DIGITAL

8, 10 & 12 BIT RESOLUTION



CONNECTS TO RS-232

ADC-16 A/D CONVERTER* (16 **channel/8 bit).$ 99.95**
ADC-8G A/D **CONVERTER* (8 channel/10 bit).$124.90**
Input voltage, amperage, pressure, energy usage, joysticks and a wide variety of other types of analog signals. RS-422/RS-485 available (lengths to 4,000'). Call for info on other A/D configurations and 12 bit converters (terminal block and cable sold separately).
ADC-8E TEMPERATURE INTERFACE' (8 **ch)..$ 139.95**
Includes term. block & 8 temp. sensors (-40' to 146° F).
STA-8 DIGITAL INTERFACE' (8 channel) $99.1
Input on/off status of relays, switches, HVAC equipment. security devices, smoke detectors, and other devices.
STA-8D TOUCH TONE INTERFACE* . . . . $ 134.90
Allows callers to select control functions from any phone.
PS-4 PORT SELECTOR (4 channels **RS-422)....$** 79.95
Converts an RS-232 port into 4 selectable W-422 ports.
CO-485 (RS-232 to **RS-422/RS-485 converter)......$ 44.95**

***EXPANDABLE...**expand your interface to control and monitor up to 512 relays, up to 576 digital inputs, up to 128 analog inputs or up to 126 temperature inputs using the PS-4. EX-16, ST-32 & AD-16 expansion cards.

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

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

* CONNECTS TO RS-232, RS-422 or **RS-485...**use with IBM and compatibles, Mac and most computers. All standard baud rates and protocols (50 to 19,200 baud). Use our 800 number to order FREE INFORMATION PACKET. Technical information (614) 464.4470.

24 HOUR ORDER LINE (800) **842-7714**
Visa-Mastercard-American Express-COD

International & Domestic FAX (614) 464-9656
Use for Information, technical support & orders.

ELECTRONIC ENERGY CONTROL, INC.
360 South Fifth Street, Suite 604
Columbus, Ohio 43215-5438

#107

ports. Don't confuse the address of LPT1 with that of port 1. They may not be the same. The operating system sets the address of LPT1 equal to the first parallel port found, which may or may not be port 1.

Each parallel port can be broken into four sections: status, control, input, and output. The circuitry uses every free line of the port, except the interrupt to the processor, which must remain off (control port D4 = 0). Depending on how your system is configured, you may end up with unexpected results or a system crash if you forget this.

The software used to read the ADC is shown in Listing 1. To start the conversion, we drive D2 of the control port low then high pulse the WR pin. After giving the ADC time to digitize the voltage, the data can be read.

The port's hardware wasn't designed for this, so to get the data, we have to do a little manipulation. First, bits O-2 are read from Status A via data bits 0, 1, and 3. There is a reason

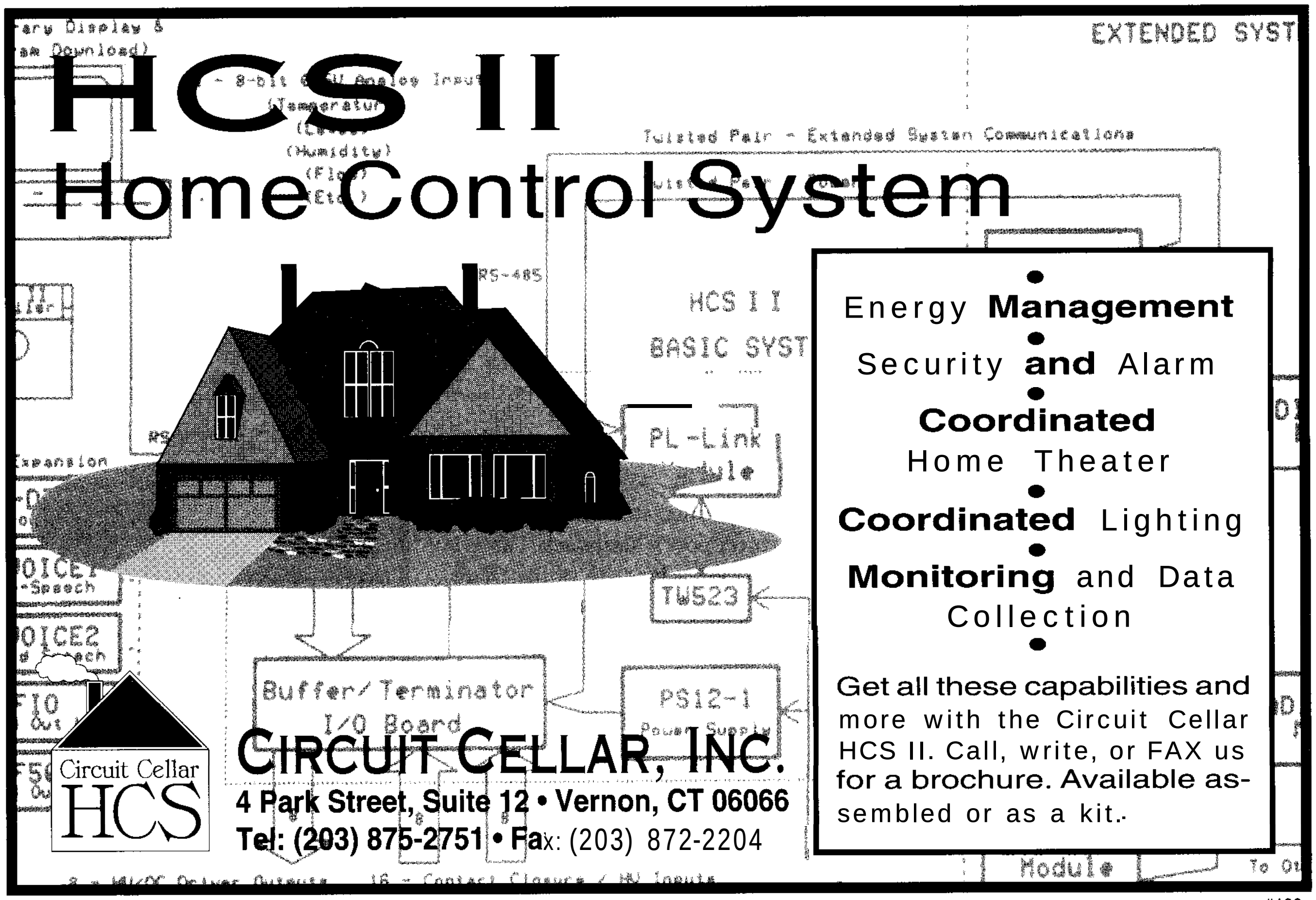
Listing I--This *procedure reads the ADC and returns the value read in the AX register. The equates are used in all three listings.*

```
Port            equ   0378h     base address of port to use
                                 change this if different
StatusPort      equ   Port+1    addr of status port for rd
ControlPort     equ   Port+2    addr of ctrl port
Dos             equ   021h      DOS interrupt
Screen          equ   010h      BIOS int for screen access
GetDisplay      equ   0fh       get video mode
Cls             equ   00        set video mode
Display         equ   02        DOS display char function
PositionCursor  equ   02        BIOS set curser function
Exit            equ   020h      program exit
GetKey          equ   08        func to get char from keybd
KeyPressed      equ   0ffh      key was pressed
CheckKey        equ   0bh       function to check key pressed
LoopsToWait     equ   0         each loop takes 17 clocks. To
                                 find number required for your
                                 system, Loops = 115 µs/time
                                 period of PC clock/17 (e.g.,
                                 a 33-MHz PC needs 224 loops)

GetADValue      proc  near
                mov   dx,ControlPort
                mov   ax,04
                out   dx,al         ;set start to 1 just in case
                xchg  ah,al
                out   dx,al         ;start to 0
                xchg  ah,al
                out   dx,al         ;then back to 1
```

(continued)

HCS II
Home Control System



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

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

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

#108

Listing 1—*continued*

```
            mov     cx,LoopsToWait  ;number of loops for 115 µs
WaitOnAd:
            loop    WaitOnAd
            in      al,dx           ;get the first 3 bits
            not     al              ;complement all the bits
            mov     ah,al
            and     ah,07           ;unused bits to 0
            shr     al,1            ;D3-D2 position
            and     al,04           ;mask all but D2
            or      ah,al           ;D0-D2 in ah
            mov     dx,StatusPort
            in      al,dx           ;get D3-D7
            and     al,0f8h         ;dump D0-D2
            or      al,ah           ;OR in D0-D2
            test    al,080h         ;top bit a one?
            jnz     AndAZero        ;no
            or      al,080h         ;invert top bit
            ret
AndAZero:
            and     al,07fh         ;invert top bit
            ret
GetADValue  endp
```

behind using D3 instead of D2. Notice that the clear input of the control port is tied to the PC's reset line. On powerup, pins 1, 14, and **17** are pulled high, pin 16 is driven low. The outputs are open collector. Thus, by using pin 17, we eliminate the possibility of bus contention with data bit 2 of the ADC.

The other five bits, D3–D7, are read at Status B. Notice bits DO-D2 and D7 are inverted by the port's hardware and must be reinverted to be used. A couple of lines of code later, we're back to the original byte.

BITS TO VOLTAGE

After the value from the ADC is processed by the fuzzy system, the DAC in Figure 4 converts it back to a value we can use. The DAC0832 is a double-buffered, multiplying DAC with complementary current outputs that is easily interfaced to many microprocessors.

The 8-bit value to be converted is simply sent to the DAC's data lines. The resulting current is presented on the I_{OUT1} pin. External conditioning circuitry then converts the current to a

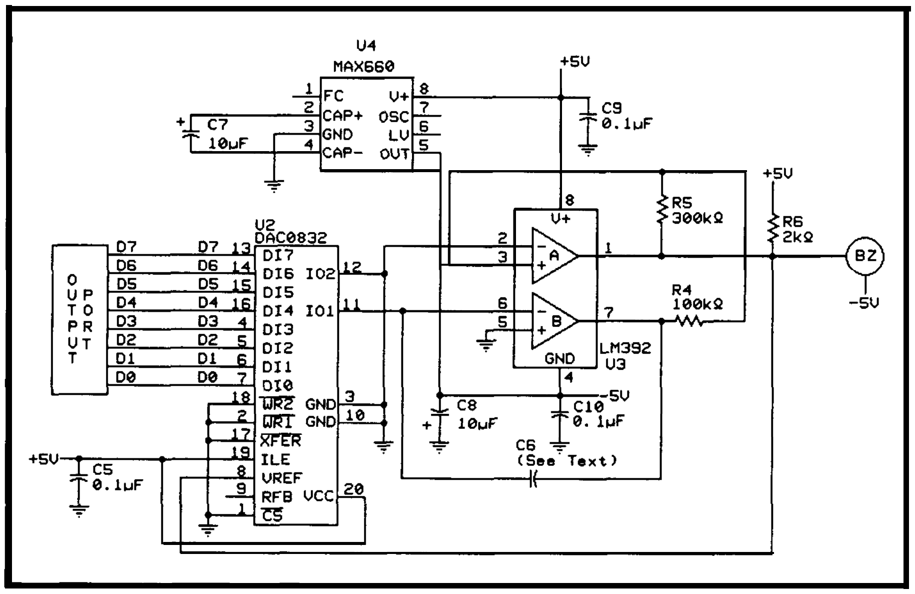


Figure 4—*After the ADC's value is processed by the fuzzy system, a DAC0832 and associated circuitry drive a piezoelectric buzzer. The buzzer's tone varies with the room temperature.*

NEW! A Complete Family of Data **Acquisitio** Products from

American Eagle **Technolog**

- Data acquisition rates from 100kHz to 1 MHz.
- Newest designs incorporate the latest technology: FIFO buffers, dual-DMA, REP INSW data transfers, programmable gains, etc.
- Simultaneous sampling option for all boards.
- Lowest prices on the market for comparable performance.
- Digital I/O w/ on-board relays.
- 16-bit analog output (D/A).
- Large inventory of popular items for next day delivery.
- Also available: signal conditioning, multi-port serial boards, device programmers, logic analyzers, & much more.

Only American Eagle Technology
Gives You All This Free Software:

- *WaveView* menu-driven software
- *Discstream* high-speed streaming software
- Complete software developers' kits for both DOS and Windows
- Drivers for all popular application programs **such as** LabView, LabWindows, NOTEBOOK, SnapMaster and DASyLab

American AET Eagle Technology
526 Durham Rd
Madison, CT 06443
Call: (203) 245-6133
Fax: (203) 245-6233

Send for Your FREE Catalog Today!

#109

voltage if necessary. The current at the output can also be affected by using the V_{REF} input for feedback.

Software to set the DAC is shown in Listing 2. The byte is written to the output port and immediately converted to a DC current.

CURRENT TO SOUND

The remainder of the circuitry in Figure 4 converts the current from the DAC to a pulse train that drives a buzzer. To stay with one supply, an ICL7660 (U4) is used to generate the -5 V. Internally, an onboard oscillator controls the on time of four MOS switches.

The first half of the cycle is used to charge C7 to Vc,. During the next half, the positive side of C7 is tied to ground and the negative, to pin 5. The charge is transferred in the opposite polarity. U4 can deliver 100 mA without significant change in the output voltage.

First, we integrate the current, creating a sawtooth waveform (U3a). The frequency of the signal is dependent on the amount of current flowing through pin 11. With an input of 0, essentially no current flows through I_{OUT1}. Thus, the frequency is 0. As the input value increases, the current also increases, varying the charge and discharge rate of C6.

The signal is passed to an open-collector comparator (U3b) to generate the square wave. Notice its output is tied to V_{REF}. With each cycle, the charge across C6 must be reversed. To do this, we drive U2/8 to +5 V and then to -5 V.

For example, assume the comparator's output just switched to +5 V. C6 was charged to +1.667 V during the previous half cycle. C6 discharges into I,,,, (electron flow) through the resistor ladder to V_{REF}. U3b's output switches to -5 V when the voltage of C6 exceeds one-third of the positive supply. At this point, all of the available supply voltage is dropped across R5. When U3b/3 is driven below U3b/2, the output switches, reversing the direction C6 charges.

To find component values, use the following formula:

Listing 2-*All that's required to send a value to the DAC is a couple of lines of code.*

```
WriteDac    proc    near
            mov     dx,Port
            out     dx,al        ;Assumes value is passed in AL
WriteDac    endp
```

Listing 3—*The minimum software required to calibrate and verify the hardware.*

```
cal      segment
main     proc     far
         assume   cs:cal
         org      0100h

start:
         mov      dx,ControlPort
         xor      ax,ax
         out      dx,al           ;set IREQ to off
         call     Clear-Screen
         mov      dl,'1'          ;indicate step 1
         mov      ah,Display
         int      DOS             ;send 1 to display
         mov      ah,GetKey
         int      DOS             ;apply power then press return
         call     Cleat-Screen
         mov      dl,'2'          ;indicate step two
         mov      ah,Display
         int      DOS             ;send 2 to the display
         call     ReadAD
         call     Cleat-Screen
         mov      dl,'3'          ;indicate step three
         mov      ah,Display
         int      DOS             ;send 3 to the display
         call     LoopWriteDac
         call     ClearScreen
         mov      dl,'4'          ;indicate step four
         mov      ah,Display
         int      DOS             ;send 4 to the display
         mov      dx,Port
         mov      al,0ffh
         out      dx,al           ;send an ffh to DAC
         mov      ah,GetKey
         int      DOS             ;get a key
         call     ClearScreen
         int      Exit            ;all done
mair     endp

ReadAD   proc     near
         mov      ah,GetDisplay   ;read display mode
         int      Screen
         mov      ah,PositionCursor ;going to set cursor to start
         mov      dh,1            ;line number
         mov      dl,0            ;column number
         int      Screen          ;bh has display page
         call     GetADValue      ;rets al with value
         mov      ah,al
         mov      cx,4            ;set up for rotates below
         ror      ah,cl           ;high nybble to low
         xchg     al,ah           ;digits right to display
         push     ax              ;save for second digit
         call     ToAscii
         mov      dl ,al
         mov      ah,Display
         int      DOS             ;char in dl to screen
         pop      ax
```

(continued)

Listing 3-*continued*

```
            xchg     al,ah
            call     ToAscii
            mov      dl,al                 next digit
            mov      ah,Display
            int      DOS
            mov      ah,CheckKey
            int      DOS                   check for key
            cmp      al,KeyPressed
            jne      ReadAD
            mov      ah,GetKey
            int      DOS                   clear keybd
            ret
ReadAD      endp

LoopWriteDac proc near
            inc      al                    next value
            mov      dx,Port
            out      dx,al
            mov      cx,08000h
WaitABit:
            loop     WaitABit              wait so change can be seen
            push     ax
            mov      ah,CheckKey
            int      DOS                   check for key
            cmp      al,KeyPressed
            pop      ax
            jne      LoopWriteDac
            mov      ah,GetKey
            int      DOS                  ;clear keybd
```

(*continued*)

$$\text{Frequency}=\frac{\text{Digital Input}}{256 \times 15\ \text{k}\Omega\ \text{(typical)} \times C6}$$

where R5 = 3 x R4. For the circuit shown, the component values are:

$$\text{Frequency max} = \frac{255}{256 \times 15\ \text{k}\Omega \times 5\ \text{nF}} \approx 13\ \text{kHz}$$

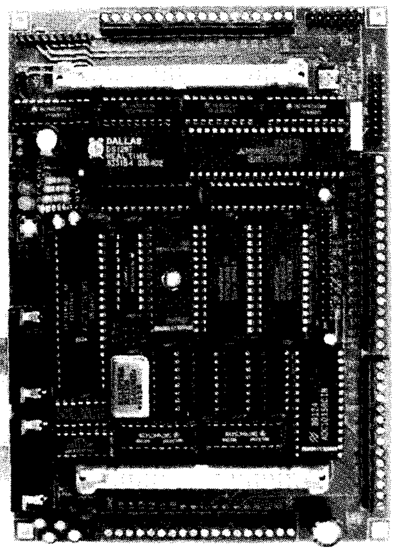
BUILDING AND CALIBRATION

Due to the simplicity and purpose of the circuitry, several ADC and DAC characteristics were not mentioned. All converters have a list of pitfalls. If your application requires increased accuracy, look into the converter's specifications before beginning.

When building the board, keep the converters' components as close as possible to the ICs. Keep the digital and analog grounds separated, except at the point of connection where they enter the board. To overlook the problems associated with driving cables, the board uses an onboard connector. If you use a cable, add the necessary components. To avoid the possibility of setting up a ground loop,

8051 EMBEDDED CONTROLLERS

RIGEL Corporation builds and supports professional development tools for embedded controller systems, with hardware and software for industrial applications, and books and kits for educational and training purposes.



THE RIGEL DIFFERENCE
All of our boards come standard with:
- 32K EPROM
- 32K RAM or EEPROM
- Machine screw sockets
- Power on LED
- All system signals on headers
- All I/O available on terminal blocks
- Sample programs
- Circuit diagrams
- IDS software READS
  READS is a DOS based menu driven, windowing platform. You can write, assemble. download. debug. and run applications software in the MCS-51 language.
- BBS Tech support

Our entire line of 8051 boards are programmable in Assembly, BASIC, "C", and Forth. We also offer low cost **Fuzzy Logic Software** for the 8051

Complete systems start at $85

CALL TODAY FOR MORE INFORMATION ON OUR PRODUCTS

RIGEL Corporation,

PO BOX 90040, GAINESVILLE, FL 32607

Telephone (904) 373-4629 BBS (904) 377-4435

#110

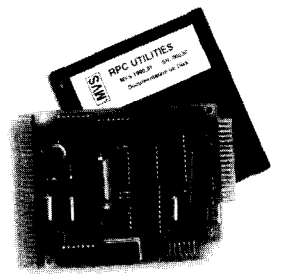
REMOTE POWER CARD!

3 VERSIONS:

RESET
WATCHDOG RESETS PC IF IT HANGS FOR HARDWARE OR SOFTWARE REASONS

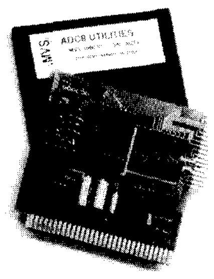
PHONE
TURN ON PC WITH PHONE, SHARE VOICE / MODEM LINE, CONTROL AC APPLIANCES

TIMER
WAKEUP OR SHUTDOWN PC, LATE NITE BACKUP / MODEM, CONTROL AC APPLIANCES



**95$** 27$ OEM

ALL MVS CARDS INCLUDE ASM, BASIC, AND C SOURCE FOR PC OR SBC



8 CHAN ADC
DATA AQUISITION, SERVO CTL, AUDIO
8-BIT RESOLUTION 22KHZ SAMPLE RATE
SHARP CUTOFF ANTI-ALIAS FILTER
CREATE STEREO BLASTER (.VOC) FILES
95$

2 CHAN DAC
VOICE MAIL, MUSIC, ALARMS, CTL VOLT
8-BIT RESOLUTION 44KHZ SAMPLE RATE
PLAYS MONO / STEREO BLASTER FILES
FUNCTIONS AS DIGITAL ATTENUATOR TOO
75$

MVS
MVS BOX 850
MERRIMACK NH
(508) 792 9507

5 YEAR LIMITED WARRANTY
FREE SHIPPING IN USA

#111

float the supply's ground (i.e., disconnect the jumper between the supply and earth ground).

The software in Listing 3 checks your setup and gives the step number and data.

Step 1-The port's interrupt request is disabled. Apply power. Verify U4/5 is approximately -5 V.
Step 2-The ADC is read and the value displayed. Change the sensor's temperature to the lowest that will be encountered. Measure the voltage on pin 6. Adjust TempMin (pin 7) for the same value. Change the sensor's temperature to maximum. Again, measure the voltage on pin 6. Set the voltage on pin 9 to one-half that of pin 6. Verify the data displayed is FF hex. Put the sensor in minimum temperature. Ensure the displayed value is 0.
Step 3—0–FF is written to the DAC in a continuous loop. Verify all data pins of the DAC are pulsing.
Step 4—FF is sent to the DAC. Check that the sawtooth waveform and the pulse train are present.

Listing 3-continued

```
          ret
LoopWriteDac endp

ToAscii  proc     near
          and     al,0fh           ;only low nybble
          add     al,030h
          cmp     al,03ah          ;> 9
          jl      ToAsciiEnds
          add     al,07h           ;get to A
ToAsciiEnds:
          ret
ToAscii  endp

ClearScreen proc  near
          mov     ah,GetDisplay
          int     Screen            ;get display mode
          mov     ah,Cls
          int     Screen            ;clear screen
          ret
Clear-Screen  endp
cal       ends
          end     start
```

ON TO FUZZY LOGIC

Now that the hardware is in place, there are several ways the control system could be generated without using fuzzy logic. One way might be to equate the temperature of the room with the different frequencies produced by the speaker.

That's not too hard. All we need to do is specify in absolute terms the difference between warm and cold. We'll say that the room is warm when the value read at the ADC is between 2.5 and 75. So, if the value is below 25, the room is considered cold.

But, is that correct? Does the room suddenly change from warm to cold at one specific temperature? What's needed is another set, which we can call KindaWarm. This set is defined as the span of inputs somewhere between Warm and Cold.

The difficulty of the task quickly escalates into a full-blown project. What's needed is a way to tell a computer that Warm, KindaWarm, and Cold are not separate entities, but a combination of all three.

Let's see how fuzzy does it.

START AT THE BEGINNING

The first thing to do is to decide what the input versus the output graph should look like. You could specify that as the room temperature increases, the frequency of the speaker also increases. In other words, the control system is the equivalent of reading the ADC and sending the value to the DAC, which is no fun at all.

Instead, specify the frequency of the speaker as maximum when the room temperature is perfect and minimum at either extreme. You should further stipulate that the graph resembles a bell curve.

SET UP THE GRAPHS

Both the input and output require a separate graph. Figure 5 shows the graph for the input. The values for the x-axis are the input values from the ADC. The values start at 0 (room coldest) and end at 255 (room warmest].

The y-axis is called the *degree of membership* (fuzzy value). Most documentation specifies its range as 0.0 to 1 .O and annotates it as μ. Instead of adding to the complexity of this, I'll refer to this axis as the *percentage of membership.*

The range starts at 0 (i.e., no membership) and ends at 100% (complete membership). If we were using Boolean logic, there would only be two values on this axis: 0 (false) and 100 (true). Fuzzy logic includes

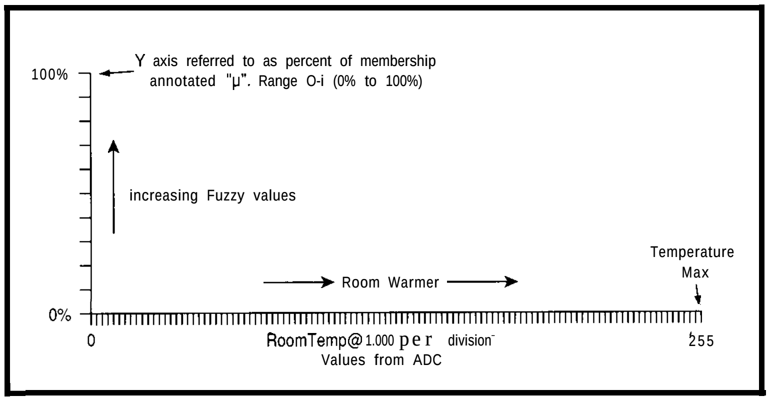


Figure 5—*After deciding what output shape is needed, input and output graphs are set up. The values on the y-axis are fuzzy values. The x-axis inputs from the ADC.*

Boolean values as well as all the other values in between the Boolean poles. One could view Boolean as a pulse train (True or False) and fuzzy as a sine wave with varying degrees of truth.

ADD THE SHAPES

Shapes replace all the drudgery of defining and redefining values. The values contained in each set are still defined, but they conform to a shape rather than matching specific numbers, Each shape defines what values are contained in the fuzzy set and the weight (i.e., percentage of membership) of each.

As an analogy, say you're in a boat and the dock is one mile away. The input shape (set) contains the distance between you and the dock. The output shape equals the range of the gas lever. When the dock is a mile away, the distance has little membership in the input set, so the lever remains unchanged. The closer you get to the dock, however, the higher the dock's membership within the shape. This membership is transferred to the output's shape, thereby decreasing the lever's position.

The shapes available are trapezoids and triangles. And, it is not readily apparent how to achieve a bell curve using these shapes. However, when rules are added, any output shape can be represented.

To select input shapes, first establish what values belong in each set. This defines the base of the shape. The left bottom is the minimum and the right, the maximum. Next, you need to decide when the fuzzy set becomes 100% true.

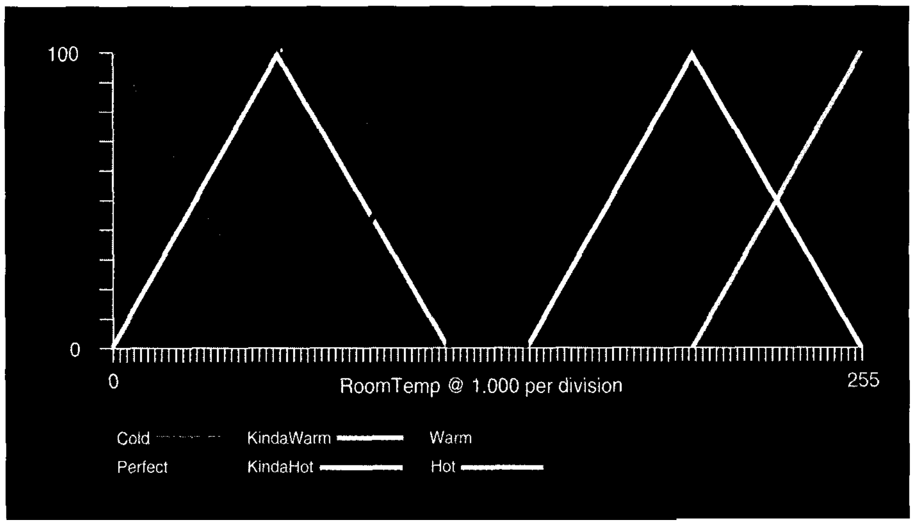


Figure *l*-*Trapezoids, triangles, and rectangles are used to define* which *ADC values are* contained *in each set and to* what degree *(i.e.,* the percentage of membership) each belongs.

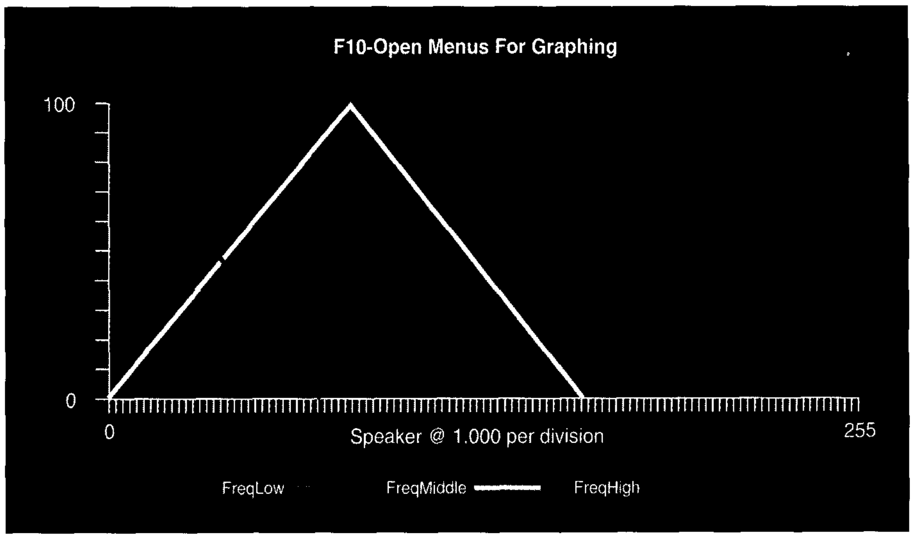


Figure **7**—*Three shapes on* ***the output*** *graph generate the be// curve.* ***FreqLow*** *controls fhe leading and falling edges,* ***FreqMiddle,*** *the transition* ***to*** *the peak, and* ***FreqHigh,*** *the peak.*

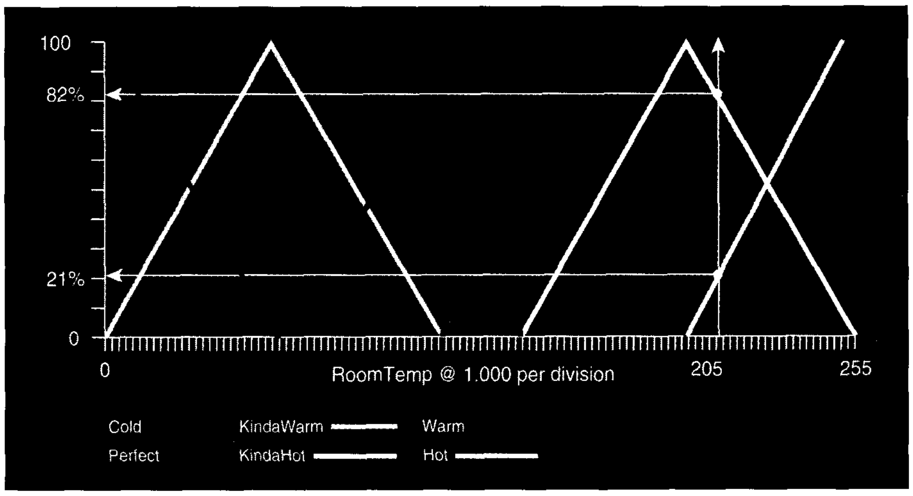


Figure **8**—*The* input from fhe ADC is converted *to* a fuzzy value by first drawing a vertical line from *the* x-axis *Horizontal lines to the y-axis determine fhe value.*

For example, refer to the Warm shape in Figure 6. Here, the room is completely warm (i.e., membership = 100%) at only one point, the shape's peak. If necessary, we could replace the triangle with a trapezoid, allowing the set to be completely true for more than one input or, for that matter, it could be true for all inputs as in rectangles. Now that the points are defined, all that's left is to connect the dots.

Unlike criteria in an absolute system, fuzzy logic lets you overlap shapes anytime values belong to more than one set. When the input value for the RoomTemp graph (Figure 6) is within the KindaWarm shape three components-Cold, KindaWarm, and Warm-interact. KindaWarm is *not* a singular set of values, but a varying degree of three.

Output shapes are a little different. Instead of working directly with the edges, centroids are used. To use an analogy again, you could consider this the center of gravity or the fulcrum on which the pivot balances. Each centroid has an x and y coordinate of which only the x value is actually used.

Shape selection is easier than inputs. You start out by finding the number of shapes required, a decision dictated by the number of dissimilar areas you're attempting to control. In this system, there are three areas and thus three shapes. FreqLow (Figure 7) controls the leading and falling edges, FreqMiddle depicts the area between FreqLow and the peak, and FreqHigh represents the peak.

Next, you determine what the output should do as fuzzy values vary. For example, the FreqLow shape is drawn so that as fuzzy values decrease, the output increases. FreqHigh is the opposite—decreasing the fuzzy values decreases the output. An equilateral triangle is used for FreqMiddle, so the output for all fuzzy values is the same, thereby creating a line. You should position the shape by visualizing the centroid with a fuzzy value of 100%. You then place it on the graph so the point is at the desired output if it were the only shape used.

With output shapes, you should always remember:

- overlapping shapes has no effect. As long as the centroid is in the same place, the base of the FreqMiddle shape can be reduced without affecting the output shape.
- unlike inputs, gaps between shapes have no effect.
- as long as their bases are identical, a centroid's *x* coordinate is the same for equilateral triangles, trapezoids with equal slopes, and rectangles, and it is always in the middle of the shape.
- when using trapezoids, the length of the top determines how close each centroid is to the previous. With a longer top, there is less change in the x coordinate of the centroids.
- of all the components, output shapes have the least affect.

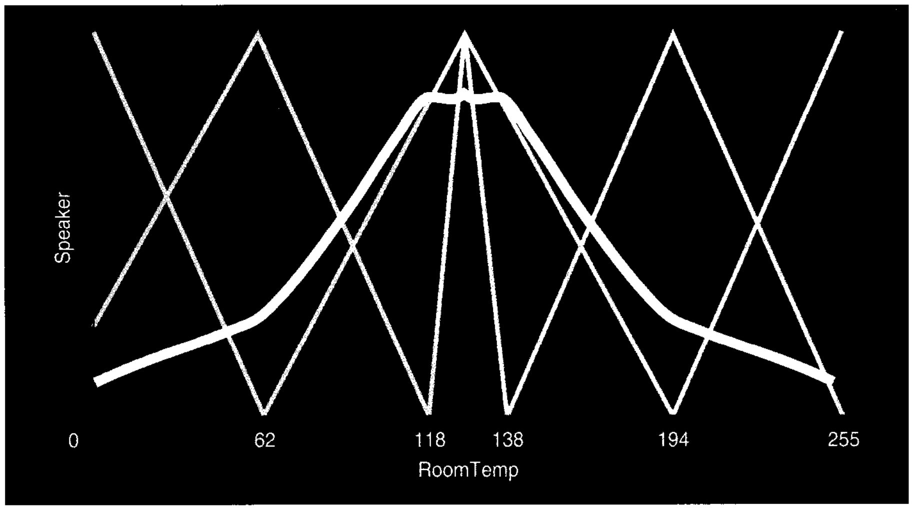


Figure 9-*Very* seldom is the *output-versus-input graph correct the first time. Three* transitions *need to be* corrected: *62,* 118-138, *and 194.*

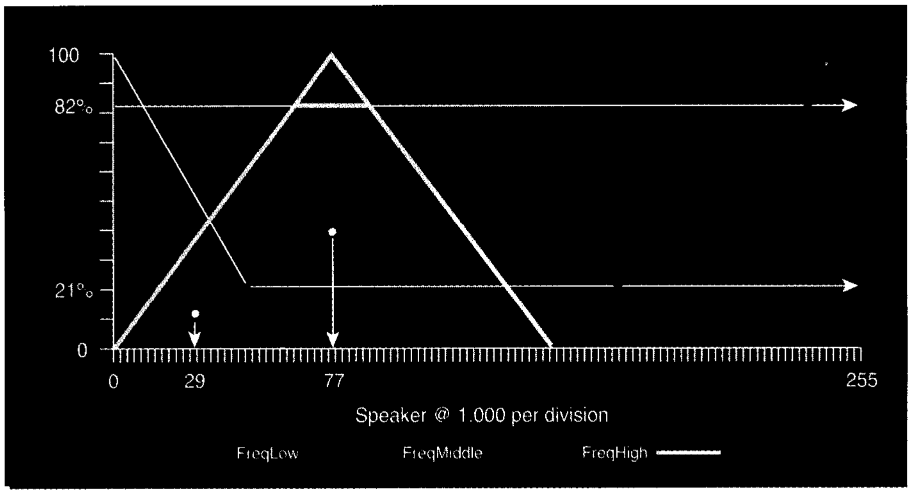


Figure **10**—*After finding the fuzzy values, they are processed by the rules and transferred to the output graph. Centroids are used to find the X-axis values.*

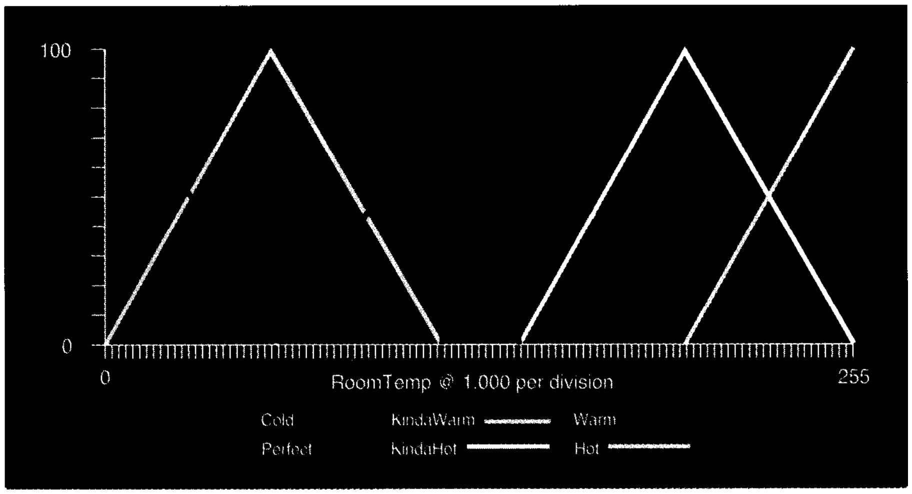


Figure 1 1—*Small modifications to the Kinda* Warm *and KindaHot shapes are all that's required to smooth the transition at the top of the bell curve.*

INPUT TO FUZZY VALUE

Figure 8 shows the completed graph for RoomTemp. To see how the fuzzy value for a given input is derived, I use a sample input value of 205.

I first draw a vertical line from 205 to the top of the graph. This line intersects two shapes. Thus, the value has membership in both of the fuzzy sets Hot and KindaHot. To find the fuzzy value, I draw a horizontal line from the point of intersection to the *y*-axis. This tells me that 205 has an 82% membership in the KindaHot set and 21% membership in the Hot set.

Once I have the fuzzy value, I can apply it to the output. Before doing that, I need to specify how the conversion takes place.

FORMULATING RULES

Rules define the relationship between input and output shapes. They are formulated much like the BASIC statement IF.. .THEN with a few modifications. Valid operators are AND (intersection], OR (union), and NOT (complement).

As stated, rules tell the system how the inputs and outputs are linked together. For instance, take the propeller of a boat. If there's no propeller (rules], the motor (input) can run all day, but we ain't goin' nowhere (output). If we install a small propeller, the rule becomes: IF motor = fast THEN output = slow. However, with a larger propeller, the rule becomes: IF motor = slow THEN output = fast.

Fuzzy rules differ from their conventional counterparts. Instead of a rule evaluating to one of two values, which are true or false, they can have varying degrees of each. For example, the statement, IF *p* AND *q* THEN *r*, evaluates to true (the rule fires) as long as both *p* and *q* have values greater than 0. Each rule specifies what input conditions must be met for the rule to evaluate to true, and once it's true, which output shapes to use.

Once the operators are understood, rule development is straightforward. Write the rules as if you're explaining how the system works. Visualize the input graph superimposed over the final output graph (Figure 9], and for each input shape, define what the output should do when the input is within the set.

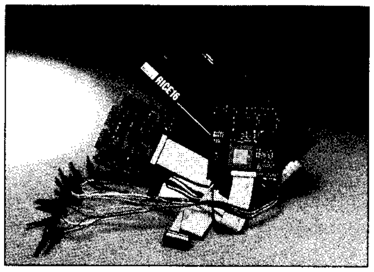
PIC16C5x/16Cxx Real-time Emulators

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

* Suggested Retail for U.S. only

RICE16 Features:

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



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

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

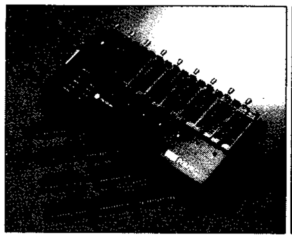
RICE-xx Junior series

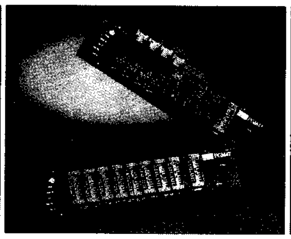
RICE-xx "Junior" series emulators support PIC16C5x family, PIC16C71, PIC16C84 or PIC16C64. They offer the same real-time features of RICE16 with the respective probe cards less real-time trace capture. Price starts at $599.

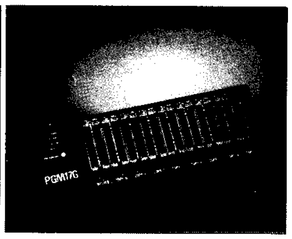
PIC Gang Programmers

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

■ Stand-alone COPY mode from a master device ■ PC-hosted mode for single unit programming ■ High throughput ■ Checksum verification on master device ■ Code protection ■ Verify at 4.5V and 5.5V ■ Each program cycle includes blank check, program and verify eight devices . Price5 start at **$599**


PGM16G: for 16C5x family


PGM47: for 16C71/84



PGM17G: for 17C42

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

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

TransdAtA Advanced **Transdata** Corporation Tel **(214) 980-2960**
14330 Midway Road, Suite 1'28. Dallas, Texas 75244 Fax (214) 980-2937

#112

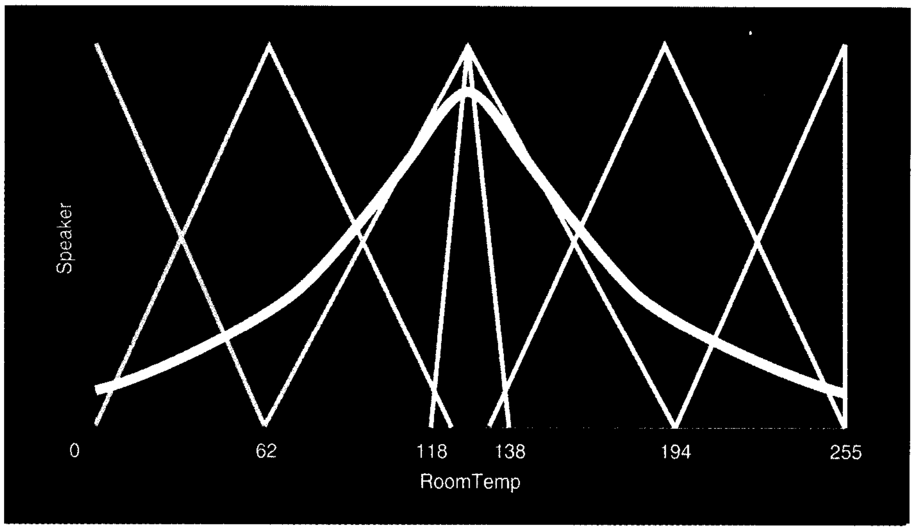


Figure 1 2—*After a couple of simple* modifications *to* the rules and shapes, *the output* is much *closer, buf still unsatisfactory. A little more work on fhe raising and falling edges, and we'// achieve our goal.*

For example, rule B in Table 2a declares that if the input falls in either the Cold or Hot shape, the frequency is set low. This rule controls the outside edges of the bell curve.

How many rules are required? It depends on the number of inputs. This system has one input with six shapes. To cover all possible input conditions, a minimum of six rules is required. If the system had two inputs, each with eight shapes, then you need 8 x 8 or 64 rules to cover all input combinations.

In addition, you might add rules to optimize the output shape. For example, this system uses ten rules instead of the minimum of six (the OR operator is used as a connector). As you can imagine, there's a point when the number becomes unmanageable. Typically, for systems with more than one input, not all combinations are possible. Thus, not all rules are required. Also, the number of shapes can often be reduced without compromising the output.

TRANSLATING FUZZY VALUES TO USABLE OUTPUTS

Now that we have the rules (recall Table 2a), let's figure out which rules are acted on. As stated, an input of 205 has membership in both KindaHot and Hot. Both rules B and C use the shapes and the OR operator, so they evaluate to true even though there is no value for Cold or KindaWarm.

Next, we have to apply the fuzzy values to the output. In fuzzy terms, this is considered defuzzifying. To take the values from input to final output, we read the crisp input, make it fuzzy by determining its membership percentage, apply rules, defuzzify, and finally convert to a crisp output.

With an input of 205, the percentage membership for rule B is Hot at 2 1%. This value needs to be defuzzified and applied to the output. Notably, the process is more complex than converting to fuzzy values.

Refer to Figure 10. Draw a horizontal line from the 2 *1%* mark on the y-axis. The shape is specified in the rule as FreqLow. That's how the percentage membership is transferred. The fuzzy value defines the top of the area (dark outline) that's used to calculate the centroid. After finding the centroid, draw a vertical line to the x-axis. This value of 29 is the output for rule B.

If this were the only rule that fired, the speaker's frequency would be close to minimum, which is not exactly what we want. With an input of 205, the room is hot but has not reached maximum. This is where fuzzy starts to work.

Due to the overlapping of the Hot and KindaHot shapes, rule C also fires, increasing the speaker's frequency. Any time more than one rule fires, the final output is modified according to the weight of each that fires. This modification corresponds to driving a car with pressure on both the brake and the gas. The final output (i.e., the speed of the car) is dependent on the amount of pressure applied to each.

Before finding the crisp output, it is necessary to calculate the centroid for the second rule (C). Once again, start with transferring the 82% membership to the FreqMiddle shape (the shape is an equilateral triangle, so the centroid is in the middle of the base regardless of what the fuzzy value is). If you refer again to Figure 10, you can see that the centroid is 77. No other rules fired. The final output is calculated by:

$$\begin{aligned} \text{CrispOutput} &= \frac{\text{Crisp 1} \times \text{Weight 1} + \text{Crisp2} \times \text{Weight2}}{\text{Weight 1} + \text{Weight2}} \\ &= \frac{29 \times 0.21 + 77 \times 0.82}{0.21 + 0.82} \\ &= 67 \end{aligned}$$

The final output is sent to the DAC.

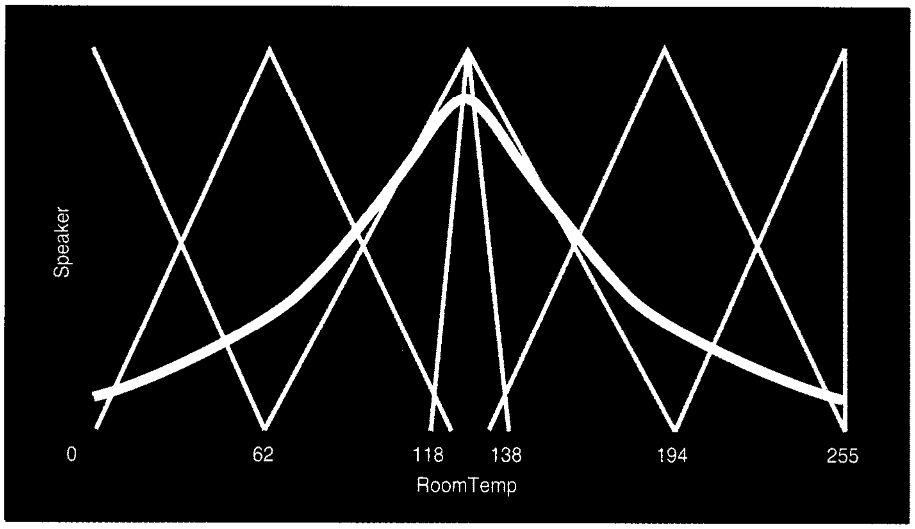


Figure 13—*The AND operator* is used to *zero in on specific areas where shapes overlap. The system is complete. Although if took several tries, it's much easier than modifying code or manipulating equations.*

```
a)  A-IF (RoomTemp Perfect) THEN (Speaker FreqHigh)
    B-IF (RoomTemp Hot) OR (RoomTemp Cold) THEN (Speaker FreqLow)
    C-IF (RoomTemp KindaWarm) OR (RoomTemp KindaHot)
         THEN (Speaker FreqMiddle)
    D-IF (RoomTemp Warm) THEN (Speaker FreqHigh)

b)  E-IF (RoomTemp Hot) OR (RoomTemp Cold) THEN (Speaker FreqLow)
    F-IF (RoomTemp KindaHot) AND (RoomTemp Hot) OR (RoomTemp KindaWarm)
         AND (RoomTemp Cold) THEN (Speaker FreqLow)
```

Table z-a) *Rules define the relationship between input and output graphs and are formulated much like the BASIC IF.. THEN statement. b) Additional rules are often required to optimize the input versus output shape.*

Notice the fuzzy value is used first in the calculation of the centroid and then again to find the final output. There are a few reasons for this. First, whatever the type of shape, centroids move very little. Using the value to define the usable area allows fine adjustments to be made on the final output shape.

Next, it must be used in the final calculation so that the relative truth of each rule is considered. For example, the weights [relative truth) of rules B and C are 21% and 82%, respectively. If we want rule C to affect the output more than rule B, we first see how true the rule is relative to itself and then how true it is relative to total truth. And, that's the basis of fuzzy logic. It doesn't deal with absolute truth, but relative truth.

Still fuzzy? Let's try it another way. Break the equation into two parts and use the first:

$$\frac{\text{Crisp 1 x Truth 1}}{\text{Truth 1 + Truth 2}}$$

Essentially, this relates the output for the rule to its truth and puts that in relation to total truth.

Why is total truth used? Because up front, there's no way to know what completely true is. In the equation above, 103% (i.e., *21% + 82%)* represents complete truth and not 100%. If ten rules fire, each with the weight of 100%, complete truth becomes 1000. In essence, regardless of what the value for total truth is, it equals 100% true.

MAKING IT WORK

The output versus the input graph for the system is shown in Figure 9. Unfortunately, as you can see, I didn't

RoomTemp	Rules Fired
IN = 0	B,E
0 < IN < 62	B,C,E,F
IN = 62	C
62 < IN 118	C,D
118 < IN < 124	A,C,D
124 IN 132	A,D
132 > IN < 138	A,C,D
138 IN<194	C,D
IN = 194	C
194 < IN < 255	B,C,E,F
IN = 255	B,E

Table **3**—*Multiple rules can evaluate to true (fire) for the same value. This prevents the absolutes common in conventional logic.*

The time is now for you to start thinking about your entry in the seventh annual Circuit Cellar Design Contest.
Entering is easy, just write or call Rose at:
Circuit Cellar Design Contest
4 Park Street, Suite 90, Vernon, CT 06066
(203) 875-2199 or fax (203) 872-2204
You'll receive an official Entry Form and a complete set of rules. All entries must be received by September 15, 1995.
You may enter as many projects as you like, but each entry must be accompanied by a separate Entry Form.

meet the objective. There are two problems: the top shouldn't be flat, nor should the corners be abrupt. I need to modify the system to achieve the desired output. Let's break the problem into three parts: the areas in the ranges of O-62, 118-138, and 194-255.

In the first area, the Cold shape begins at 0 and ends at 62. We need to add another rule that "pulls" the output toward 0 when the input is within this shape. Also, notice that the Hot shape controls the same portion, but on the opposite end. Both ends get fixed with this one rule.

The final output can be derived by two methods: the max method and the average. Using the max method, only one centroid per output shape is used in the final calculation. The average averages all centroids.

For example, to fix the problem with the abrupt corners, you can specify rule B twice. With an input of 205, three centroids would be produced: one for rule C and two for B. Using the max method, the final crisp output becomes:

$$\frac{\text{Centroid C} \times \text{Fuzzy C} + \text{Largest Centroid B} \times \text{Fuzzy B}}{\text{Fuzzy C} + \text{Fuzzy B}}$$

Using the average method, the final crisp output becomes:

$$\frac{\text{Centroid C} \times \text{Fuzzy C} + 2 \times \text{Centroid B} \times \text{Fuzzy B}}{\text{Fuzzy C} + 2 \times \text{Fuzzy B}}$$

At first glance, it may appear that specifying a rule twice doubles its effect. Unfortunately, it's not that easy. Due to the interaction between all centroid and fuzzy pairs (it's that relative truth thing), it may fix the problem, create another, or have no effect. It works here, so we use it.

How should the area between 118 and 138 be adjusted? Should we modify the rules or the shapes?

To be honest, it's a toss up. It's kinda like your toast. If you toast some bread and it's too light, you have to decide how to make it darker. You could toast it again (input), change the toaster's setting (rules), or manually hold the lever down (output). All of these activities achieve the same thing.

However, since modifying shapes is the easiest, let's start there. If you superimpose the input graph over the output, the area in question falls within the perfect shape. In fact, the top goes flat as soon as the shape comes into existence.

When the input is within this shape, two rules (A and D) fire. There's our problem. Both rules tell the system to increase the frequency. To fix it, you need to add interaction, holding the output low longer. Since rule C uses the FreqMiddle shape, it could provide the fuzziness needed if we make it fire. After modifying both the KindaWarm and KindaHot shapes as shown in Figure 11, the problem is solved. At the points of overlap, the rules "fight" each other, smoothing the transition.

The modified output is shown in Figure 12. Although it's better, it still isn't right. The transition with an input of 62 is still too abrupt. To smooth it further, add another rule that fires only when input is within the area where the Cold and Kinda-

---

C COMPILERS
CROSS ASSEMBLERS
DEBUGGERS

68HC08	8051/52
6809	8080/8085
68HC11	8086/186
68HC16	8096/196

Low Cost!! PC based cross development packages which include EVERYTHING you need to develop C and assembly language software for your choice of CPU.

- MICRO-C compiler, optimizer, and related utilities.
- Cross Assembler and related utilities.
- Hand coded (efficient ASM) standard library (source included).
- Resident monitor/debugger (source included)*
- Includes text editor, telecomm software and many other utilities.

* 68HC08 and 68HC16 kits do not include monitor/debugger.

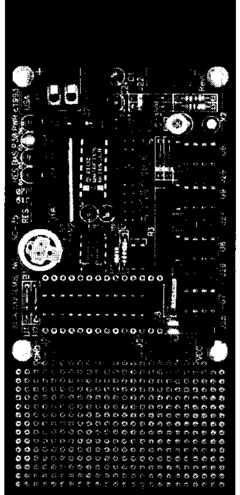
Each Kit: $99.95. + s&h (please specify CPU)

A *SPECIAL!!* *Super Developer's Kit*
Includes all 8 kits above, plus additional assemblers for 6800, 6801/6803, and 6502. ***Reg. $400.00 NOW $300.00***
(Ask for SDK Special)

DDS
Dunfield Development Systems
P.O. Box 31044 Nepean, Ont. **K2B 8S8**
CANADA
Tel/BBS: 613-256-5820 Fax: 613-256-5821

#113

8051
80C32



87C751
87C752

Use one of our embedded controllers to save time and money. **They are ideal for developing products, test fixtures and prototypes.**

We offer a complete line of controller boards and software tools for the 8051 and 87C751 families of microcontrollers. Complete packages are available to help you develop your projects.

Features:

- **Breadboard area**
- **Flexible I/O arrangement**
- **Powerful controller BASIC for the 87C752 or 80C32**

Ph: (702) 831-6302
Fax: (702) 831-4629

Iota Systems, Inc.
POB 8987 • Incline Village, NV 89452-8987

#114

Warm shapes overlap. The fuzzy value applies to the FreqLow shape:

IF (RoomTemp Cold) AND
  (RoomTemp KindaWarm)
THEN (Speaker FreqLow)

The AND operator allows us to zero in on specific areas of shape overlap. At any other time, one or more of the fuzzy values is 0, preventing the rule from firing.

The same problem exists on the falling edge of the output. Instead of writing a separate rule, we can combine both using the OR operator as shown in Table 2b, rule F.

The final output is shown in Figure 13. Table 3 shows which rules fire as the input varies. Although it took several tries to get the optimum shape and rule mix, it's much easier than modifying code or manipulating equations.

SOME USES

The applications are endless. Some of them include:

- robotics
- home and office burglar alarm and climate control
- AC/DC motor control
- weather prediction through humidity, temperature, pressure, and so on
- analyzing data for patterns and trends.
- controlling product flow in production (anything requiring control or generalization is a possible application)

WHERE TO GO FROM HERE

I hope this gives you a basic understanding of fuzzy logic. The next step is to apply the principles outlined in the article to your own designs. All you need is a lot of time, a ream or two of graph paper, and a calculator. If that is not appealing, there is a better way.

I have development systems for both the novice and advanced user. Please call or write for details. ❑

Bud Moss has worked in the electronic and electrical fields for more years than he cares to remember. After researching and using several unconventional technologies, he founded Xcentrics to provide affordable fuzzy logic development tools. He may be reached at 75313.2353@compuserve.com.

SOFTWARE

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

SOURCE

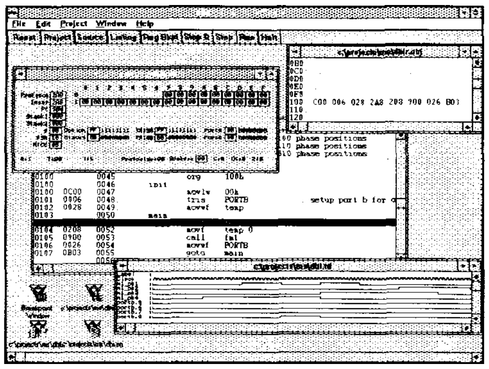
Xcentrics, Inc.
P.O. Box 1268
McMinnville, OR 97 128
(503) 434-5729

I R S

401 Very Useful
402 Moderately Useful
403 Not Useful

PIC **Interactive Development Environments for the Microchip Technology, Inc. PIC family of microcontrollers. Microsoft Windows 3.1 MDI Applications**

MicroView™ Simulation Environment



```
0B0
0C0
0D0
0E0
0F0
100  C00 006 028 2A8 208 900 026 B03
110
120
```

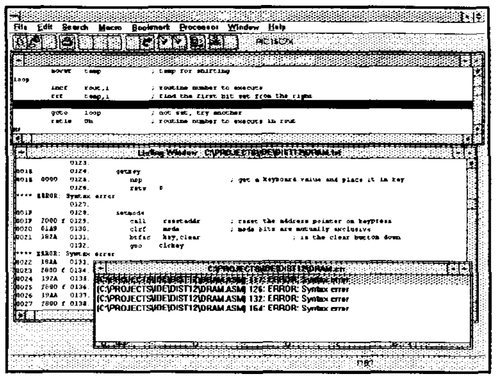
```
00 phase positions
10 phase positions
10 phase positions
0100          0045           org      100h
0100          0046      init
0100   0C00   0047           movlw    00h
0101   0006   0048.          tris     PORTB          setup port b for o
0102   0028   0049.          movwf    temp
0103          0050      main
0104   0208   0052           movf     temp 0
0105   0900   0053           call     fml
0106   0026   0054           movwf    PORTB
0107   0B03   0055           goto     main
```

MICROVIEW™ is a full-featured simulation environment for the Microchip Technology, Inc. PIC16C5X microcontrollers. Other simulation engines available.

Features:

- Integrated Programming Editor, Macro-Assembler
- Simultaneous Multiple Device Simulations
- Graphical Timing Diagrams, Input Stimulus Generation
- Step, Register, Full Break Point Capability
- Compatible with PICMASTER. and the Parallax Downloader

Assemble! ™ Universal PIC Assembler



```
        movwf   temp            ; temp for shifting
loop
        incf    rout,1          ; routine number to execute
        rrf     temp,1          ; find the first bit set from the right
        goto    loop            ; not set, try another
        retlw   0h              ; routine number to execute in rout
```

Listing Window C:\PROJECTS\IDE\DIST12\DRAM.lst

```
               0123.
0018           0124.      getkey
0018   0000    0125.          nop                    ; get a keyboard value and place it in key
               0126.          retw    0
**** ERROR: Syntax error
               0127.
001F           0128.      setmode
001F   2000 f  0129.          call    resetaddr      ; reset the address pointer on keypress
0020   0149    0130.          clrf    mode           ; mode bits are mutually exclusive
0021   182A    0131.          btfsc   key,clear                  ; is the clear button down
               0132.          goo     clrkey
**** ERROR: Syntax error
0022   182A    0133.
0023   2880 f  0134.
0024   192A    0135.
0025   2880 f  0136.
0026   192A    0137.
0027   2800 f  0138.
```

C:\PROJECTS\IDE\DIST12\DRAM.err

```
(C:\PROJECTS\IDE\DIST12\DRAM.ASM) 117: ERROR: Syntax error
(C:\PROJECTS\IDE\DIST12\DRAM.ASM) 126: ERROR: Syntax error
(C:\PROJECTS\IDE\DIST12\DRAM.ASM) 132: ERROR: Syntax error
(C:\PROJECTS\IDE\DIST12\DRAM.ASM) 164: ERROR: Syntax error
```

ASSEMBLE! is a universal macro assembler for the Microchip Technology, Inc. microcontrollers. Assemble! supports all of the Microchip device families.

Features:

- Integrated Programming Editor, Macro-Assembler
- Bookmarks, Macro Key Recording, Search and Replace
- Error Report, Listings, Context Sensitive On-Line Help
- COD File Format, INHX8S, INHX8M, INHX16
- Direct Download to PICSTART
- Compatible with PICMASTER. and the Parallax Downloader

MicroView™ $149.95USD VISA/MC **Assemble!™ $99.95USD VISA/MC**

TRISYS Send check or Money Order, VISA/MC Accepted, Company Purchased Orders Accepted, Shipping and Handling Included.

TRISYS, Inc. P.O. Box 11087 • Glendale, AZ 85318-1087 • USA • PH: (602) 581-1760 • FAX: (602) 516-0582

[PIC, PICMASTER, and PICSTART are trademarks and tradenames of Microchip Technology, Inc. , Parallax is the tradename of Parallax, Inc]

#115

FEATURE ARTICLE

Jim Sibigtroth

Fuzzy Logic for Embedded Microcontrollers

Fuzzy logic doesn't necessarily need lots of horsepower. Many embedded applications that use more traditional control schemes can benefit from the use of fuzzy logic. Jim looks at how to keep things simple and speedy.

After describing basic fuzzy-logic concepts, this article explains how to implement fuzzy-inference algorithms in a general-purpose embedded controller. The examples, written in assembly language, are for an MC68HC11, but the algorithms could be adapted for any general-purpose microcontroller. Code size is surprisingly small and execution time is fast enough to make fuzzy logic practical even in small embedded applications.

Perhaps because of its strange sounding name, fuzzy logic is still having trouble getting accepted as a serious engineering tool in the United States. In Japan and Europe, the story is quite different. The Japanese culture seems to respect ambiguity, so it is considered an honor to have a product which includes fuzzy logic. Japanese consumers understand fuzzy logic as intelligence similar to that used in human decisions.

In the US, engineers typically take the position that any control methodology without precise mathematical models is unworthy of serious consideration. In light of all the fuzzy success stories, this position is getting hard to defend.

I think the European attitude is more appropriate. It recognizes fuzzy logic as a helpful tool and uses it. They regard the difficulties of the nomenclature as a separate problem. Since the term "fuzzy" has negative connotations, they simply don't advertise that products include fuzzy logic.

NOT AS FUZZY AS IT SOUNDS

Curiously, the results produced by fuzzy-logic systems are as precise and repeatable as those produced by respected traditional methods. Instead of indicating lack of precision, the term "fuzzy" more accurately refers to the way real-world sets have gradual boundaries.

When we say "the temperature is warm," there is not a specific temperature at which this expression goes from completely false to completely true. Instead, there is a gradual or fuzzy boundary, which requires a nonbinary description of truth. In fact, the fuzzy logic definition for a set contains more information than the conventional binary definition of a set.

In conventional systems, the range of an input parameter is broken into sets that begin and end at specific values. For example, a temperature range described as warm might include the temperatures 56–84°F (see Figure la). The trouble with this thinking is that the temperature 84.01°F suddenly stops being considered warm. This abrupt change is not the way humans think of concepts like "temperature is warm."

Fuzzy logic uses a two-dimensional membership function to express the meaning of an input parameter such as "temperature is warm." Figure Ib shows how to express the meaning of warm temperature in a fuzzy-logic system. The x-axis shows the range of possible values for the input parameter temperature. The y-axis shows the degree to which temperature can be said to be warm (the degree of truth for the expression "temperature is warm"). The y-axis ranges from $00 (not at all true) to $FF (completely true). You may see references where the degree of truth varies between 0.00 and 1 .OO, but in an embedded microcontroller, it is more practical to treat the truth value as an R-bit binary value between $00 and $FF.

OVERALL STRUCTURE OF A FUZZY KERNEL

Figure 2 shows a block diagram of a fuzzy-logic inference program in an embedded controller. Preprocessed system inputs enter the top of the fuzzy-inference kernel and system outputs leave at the bottom. The three processing blocks in the fuzzy kernel are executed in series each time the fuzzy kernel is called.

For each of the three processing blocks in the fuzzy kernel, there is a corresponding data structure in the knowledge base. *Fuzzification* compares the current value of system inputs against the input membership functions to determine values for fuzzy inputs stored in S-bit RAM locations.

As rules from the rule list are processed, current fuzzy input values are used. The resulting values are stored in the fuzzy output locations in a second RAM array. Finally, the fuzzy output values are combined in the *defuzzification* step to produce system output values.

The fuzzy-inference kernel and the knowledge base can be developed independently. The advantage to this is that the microcontroller programmer developing the fuzzy kernel doesn't need to be familiar with the process to be controlled. Similarly, the process expert doesn't need to be a microcontroller programmer.

All that is necessary is that they agree on some basic ground rules such as number of inputs and outputs, number of labels for each input and output, and some basic limitations on rule structure. The fuzzy kernel can even be developed by a third party such as a semiconductor manufacturer or a fuzzy-development-tool vendor. The kernel software described in this article is an example of a fuzzy kernel developed without any detailed knowledge of the systems in which it will be used.

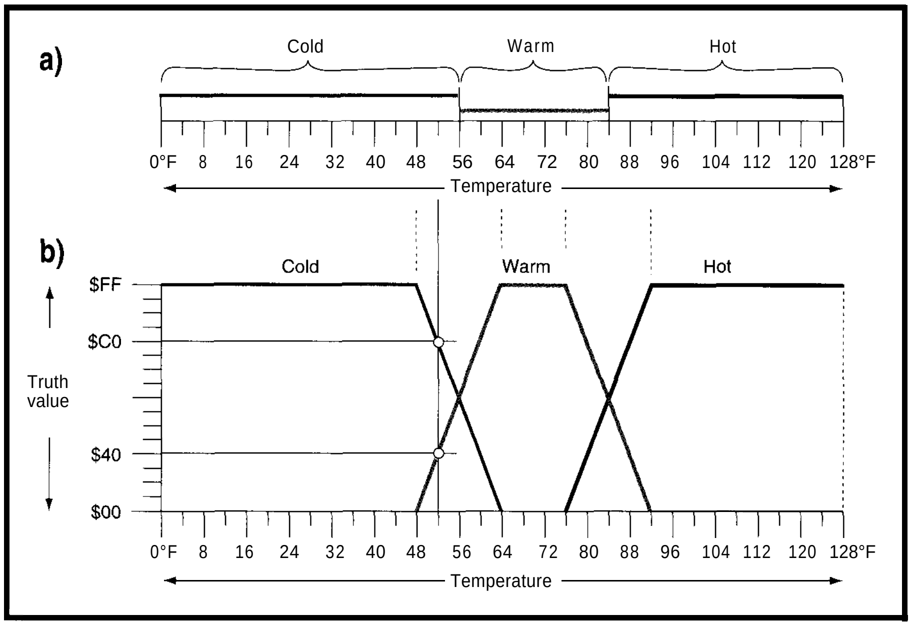


Figure 1—*Traditional sets are simply defined by their endpoints. Fuzzy sets add a second dimension to express the degree of truth (on the y-axis), which allows sets to be defined with gradual boundaries between false and true.*

EXPRESSING EXPERT KNOWLEDGE

For years, researchers have struggled to translate human knowledge into a form which can be manipulated by computers. If researchers could express the meaning of an idea like "temperature is warm" in an unambiguous numerical way, a digital computer could use this knowledge to make decisions similar to those made by competent humans. Fuzzy logic has taken a giant step in this direction with the introduction of membership *functions.*

A fuzzy logic system is programmed with a series of rules such as "If temperature is warm and pressure is medium, then heater is full_on." This natural-language control rule is simple enough for a human expert. Although conventional digital systems have trouble dealing with concepts like "temperature is warm," fuzzification gets around this problem by assigning a concrete number between $00 [false) and $FF (true) to this linguistic expression so that the microcontroller can process it further.

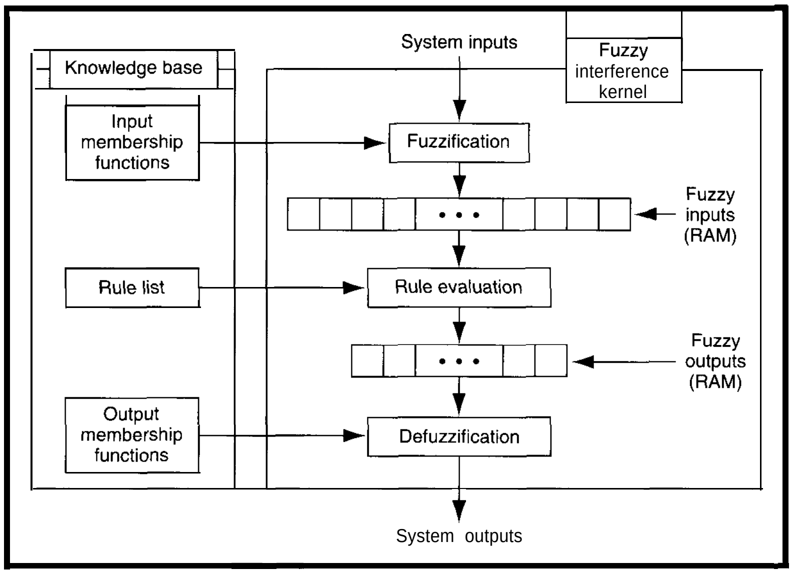


Figure 2—*The knowledge base in this block diagram is developed by an application expert and the inference kernel, by an MCU programmer. Note the relationships between data sfrucfures in the knowledge base and the three main processes in the kernel. Fuzzy-input and fuzzy-output RAM data sfrucfures hold intermediate results during execution.*

FUZZIFICATION

In this step, the current value of each input is compared to the membership functions for each label of the corresponding input. From this, it is possible to determine a numerical truth value for every label of every input. Input signals are typically preprocessed sensor signals scaled to fit in the range from $00 to $FF. Preprocessing sensor inputs is an ordinary part of any embedded-control application, and fuzzy logic does not require any special skills for this job.

The inputs to the fuzzification process are the current 8-bit value of each system input and a membership function definition for each linguistic label of each system input. Results of the fuzzification step are fuzzy inputs in RAM-there's one byte for each label of each system input.

The fuzzy kernel in this article uses trapezoidal membership functions defined by two points and two slopes per membership function in nonvolatile memory. In other words, in an application with two system inputs and five labels per input, there would be 10 membership functions (4 bytes each = 40 bytes of ROM or EEPROM) and 10 fuzzy inputs (1 byte each = 10 bytes of RAM).

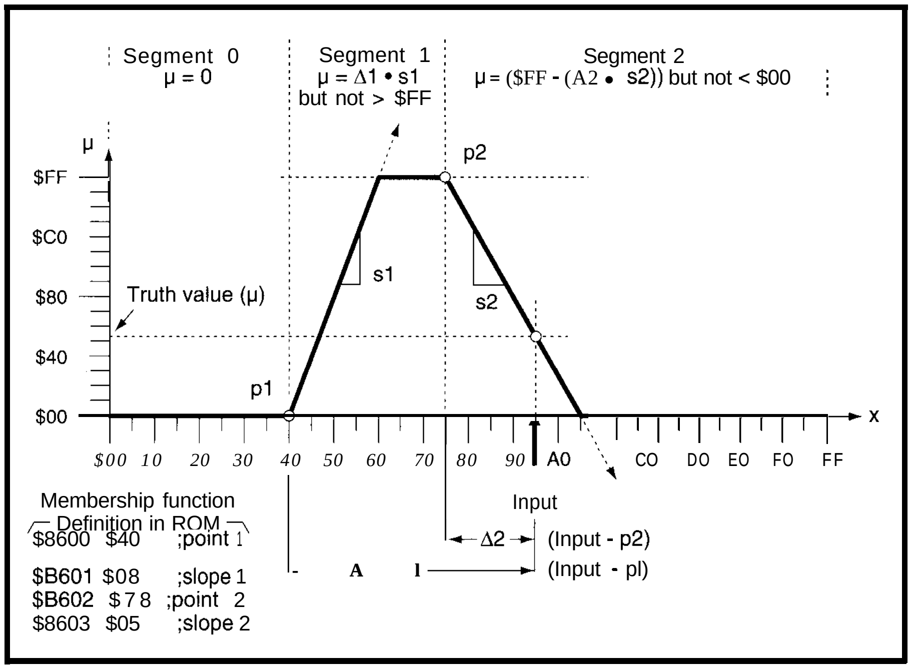


Figure 3—This figure shows one way of defining and evaluating a trapezoidal membership function. Segment 0 is defined by the position of point 1, segment 1 is defined by the values of point 1 and slope 1, and segment 2 is defined by the values of point 2 and slope 2.

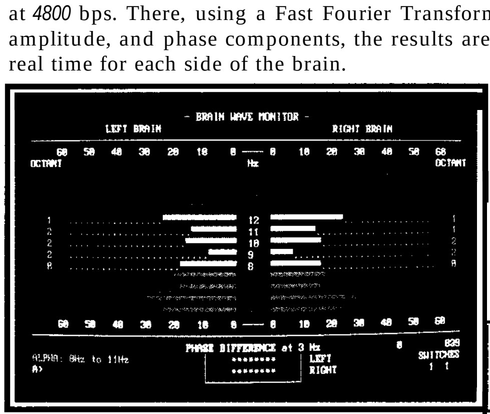
The major processing element in this step is a routine to determine the y-intercept on the membership function for one label corresponding to the current value of one system input. Place this routine inside of two concentric loops. The inner loop executes once for each label of one input.

The outer loop executes once for each system input. In a system that has two inputs with five labels each, the outer loop executes twice and the

HAL-4

EEG Biofeedback Brainwave Analyzer

The HAL-4 kit is a complete battery-operated 4-channel electroencephalograph (EEG) which measures a mere 6" x 7". HAL is sensitive enough to even distinguish different conscious states-between concentrated mental activity and pleasant daydreaming. HAL gathers all relevent alpha, beta, and theta brainwave signals within the range of 4-20 Hz and presents it in a serial digitized format that can be easily recorded or analyzed. HAL's operation is straightforward. It samples four channels of analog brainwave data 64 times per second and transmits this digitized data serially to a PC at 4800 bps. There, using a Fast Fourier Transform to determine frequncy, amplitude, and phase components, the results are graphically displayed in real time for each side of the brain.



HAL-4 KIT......NEW PACKAGE PRICE - $279 +SHIPPING

Contains HAL-4 PCB and all circuit components, source code on PC diskette, serial connection cable, and four extra sets of disposable electrodes.

to order the HAL-4 Kit or to receive a catalog,

CALL: (203) 875-2751 OR FAX: (203) 875-2204

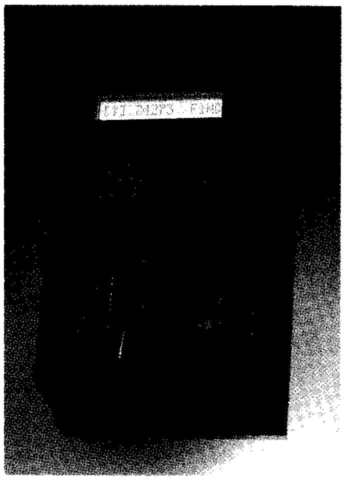
CIRCUIT CELLAR KITS • 4 PARK STREET

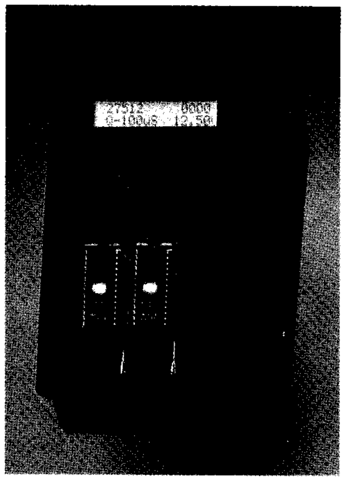
SUITE 12 • VERNON • CT 06066

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

0|1|0
1|1|1
0|1|0
DATA
Genie™

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





HT-28 *Express*

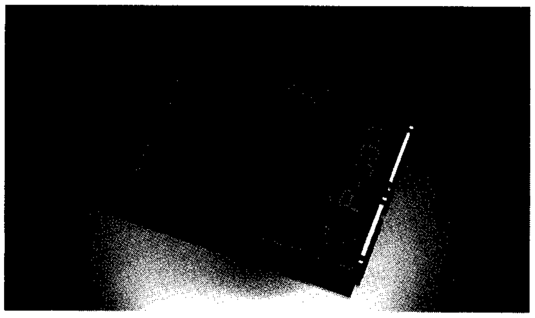
DIGITAL IC TESTER

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

HT- 14 *Express*

EPROM PROGRAMMER

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



P-300

PC INTERFACE CARD PROTECTOR

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

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

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

Data Genie products ore backed by a full 1year limited factory warranty.

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

#117

routine inside the inner loop a total of 10 times (five times for each pass through the outer loop). Figure 3 shows the routine for finding the *y*-intercept for one label of an input.

Listing 1 shows a practical algorithm for finding the grade of membership for one label of one system input. This routine is embedded inside the inner loop of the fuzzification process and follows the pattern mentioned above. The outer loop executes twice while the inner loop circles 10 times.

This routine also updates two pointers. The first one (X) points at the 4-byte membership function definition in the knowledge base. The second one (Y) points at the RAM location where the fuzzy input (result) will be stored.

The calculations associated with segment 2 take slightly longer than those for segment 1, so the routine checks to see if the input is there first. This check helps balance the execution time and keeps the worst-case path as short as possible. The shortest path occurs when the input is in segment 0. Two range-checking sequences, BLS NOT_SEG2 and BLO HA V_G RA D, are in this path.

This algorithm uses the x-positions of two points and two unsigned slope values to define a membership function. While it would be more straightforward to describe a trapezoid with four corner points, it then requires at least one divide during run time. The method described here never needs to execute anything more difficult than one 8-bit multiply to find a grade of membership.

Trapezoidal membership functions are commonly used because they meet the requirement of providing a gradual transition from false to true while requiring only simple calculations to compute an intercept. Some programs only allow triangular membership functions, but trapezoids are just as easy to process. A trapezoid with a top width of zero makes a triangular membership function.

RULE EVALUATION

Although rules sound like arbitrary, natural-language statements, they follow a fairly strict syntax. The typical fuzzy-logic kernel in a small, Listing 1—This *routine performs* fuzzification *for one label of one system input.* Refer to *Figure* 3 while *studying this program.*

```
*GET-GRADE-Routine to project a current input value onto          *
*   an associated input membership function (fuzzification). *
*   Result is stored to fuzzy input and pointers are updated. *
*ENTRY VALUES: A = Current system input value                      *
*              X = Pointer to membership function in ROM           *
*              Y = Pointer to fuzzy input in RAM                   *
*EXIT VALUES:  A unchanged (ready for next GET-GRADE call)  *
*              B used internally to calculate grade (result)*
*              X add 4 to point at next MF definition              *
*              Y add 1 to point at next fuzzy input in RAM  *

GET-GRADE  PSHA              ;Save input value of A
           CLRB              ;In case grade = 0
           SUBA 2,X          ;Input value - pt2 -> A
           BLS  NOT_SEG2     ;If input < pt2
           LDAB 3,X          ;Slope 2
           BEQ  HAV_GRAD     ;Skip if zero slope
           MUL               ;(In - ptl) * slp2 -> A:B
           TSTA              ;Check for > $FF
           BEQ  NO_FIX       ;If upper 8 = 0
           CLRB              ;Limit grade to 0
           BRA  HAV_GRAD     ;In limit region of seg 2
NO_FIX     SUBB #$FF         ;B - $FF
           NEGB              ;$FF - B
           BRA  HAV_GRAD     ;($FF -((In - pt2) * slp2))
NOT_SEG2 ADDA 2,X            ;Restore input value
           SUBA 0,X          ;Input value - ptl -> A
           BLO  HAV_GRAD     ;In < ptl so grade = 0
           LDAB 1,X          ;Slope 1
           BEQ  ZERO_SLP     ;Skip if zero slope
           MUL               ;(In - ptl) * slpl -> A:B
           TSTA              ;Check for > $FF
           BEQ  HAV_GRAD     ;Result OK in B
ZERO_SLP  LDAB #$FF          ;Limit region or zero slope
HAV_GRAD   INX               ;Point at next MF spec
           INX
           INX
           INX
           STAB 0,Y          ;Save one fuzzy input
           INY               ;Point at next fuz input
           PULA              ;Restore A register
```

embedded-control system limits rules to the following form:

IF system-input-x is label-a
    AND system-input-y is label-b
    THEN output-w is label-c.

Each of the linguistic expressions like "system-input-x is label-a" corresponds to a specific fuzzy input value in RAM. These values are determined by the fuzzification step. The expression "system_output_w is label-c" corresponds to a specific fuzzy output. AND is a fuzzy operator which corresponds to the mathematical minimum operation. All the linguistic expressions on the left side of the rule are connected by ANDs. The truth value for the whole rule is the value of the smallest fuzzy input on the left side. There is an implied OR between successive rules, which corresponds to the mathematical maximum operation.

Before processing the rules, all fuzzy outputs are initialized to $00 (meaning not true at all). As rules process, the truth value for the current rule is stored in each fuzzy output on the right side of the rule unless the fuzzy output is already bigger (this is the maximum operation).

Rules can be stored in the knowledge base as a simple list of pointers to fuzzy inputs and fuzzy outputs. For the kernel described in this article, a 7-bit offset from the start of the fuzzy input array is used for each rule antecedent.

Tight Budget '51 Emulation

8051 Family Emulator is truly Low Cost!

The DryICE Plus is a modular emulator designed to get maximum flexibility and functionality for your hard earned dollar. The common base unit supports numerous 805 1 family processor pods that are low in price. Features include: Execute to breakpoint, Line-by-Line Assembler, Disassembler, SFR access, Fill, Set and Dump Internal or External RAM and Code. Dump Registers, and more. The DryICE Plus base unit is priced at a meager $299, and most pods run only an additional $149. Pods are available to support the 8031/2, 8751/2, 80C154, 80C451, 80C535, 80C537, 80C550, 80C552/62, 80C652, 80C851, 80C320 and more. Interface through your serial port and a comm program. Call for a brochure or use INTERNET. We're at **info@hte.com** or ftp at ftp.hte.com

Tighter budget? How about *$149* emulation?

Our $149 DryICE model is what you're looking for. Not an evaluation board - much more powerful. Same features as the DryICEPlus, but limited to just the 803 1/32 processor.

You can afford it!

So, if you're still doing the U V **Waltz** (Burn-2-3. Erase-2-3). or debugging through the limited window ROM emulators give, ***call*** us **now** for relief! Our customers say our products are still the best Performance/Price emulators available!

Look into our Single Board Computer solutions, too!

HTE

HiTech Equipment Corp
9400 Activity Road
San Diego, CA 92126
[Fax: (619) 530-1458]

Since 1983

(619) 566-1 892

VISA MasterCard

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

#118

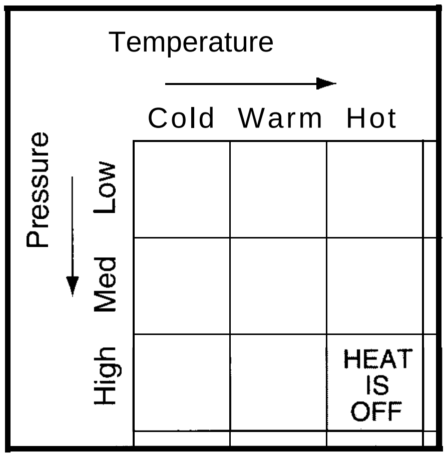


Figure 4-All possible combinations of input conditions are summarized in this course rule matrix. The shaded cell represents the rule "If temperature is hot and pressure is high, then heat is off."

The MSB of all antecedent pointers is clear. A byte with the MSB set plus a 7-bit offset from the start of the output array is used for each rule consequent.

Since the MSB distinguishes consequents from antecedents, rules may have any number of inputs or outputs. It would be faster, but less flexible, to define rules with a fixed structure such as two antecedents and one consequent. It would also be faster to use whole addresses rather than offsets in the rule list, but that would more than double the amount of memory required for the rule list.

Since each input has only a finite number of labels, there are only a certain number of possibilities for unique rules. A system with two inputs, each having three labels, has a maximum of nine possible rules as shown in Figure 4. As you can see, the treatment of values is very coarse. No transition regions are shown between adjacent labels of the inputs.

Figure 5 corrects this. It shows the membership functions below and to the right of the rule matrix. This figure shows the areas where more than one label of an input is true at the same time. The knowledge base only specifies the system-output level at the nine shaded cells of Figure 5. The other cells represent combinations of input values that cause two or four rules to be true to some degree at the same time. In these areas, the defuzzification step combines the recommended actions of all of the contributing rules.

DEFUZZIFICATION

After the rule-evaluation step, each of the fuzzy outputs has a value corresponding to the degree that output action should be applied. These can be considered as recommendations for the system-output level. The defuzzification step combines these separate recommendations into a single, composite system-output value.

The program in this article uses singleton membership functions, which are simply the x-axis position of one label of a system output. The fuzzy output value in RAM represents the height (y value] of this membership function or the degree to which it should apply. The following formula shows the calculation needed for defuzzification:

$$\frac{\sum_{i=1}^{n} F_i \times S_i}{\sum_{i=1}^{n} F_i}$$

where n is the number of fuzzy outputs associated with system output, F_i is a weight (fuzzy output value from runtime RAM), and S_i is a membership-function singleton position (from the knowledge base]. The result of this calculation is the system-output action. F_i and S_i are 8-bit values and the value of n is typically 8 or less. This makes the numerator a 19-bit value and the denominator an 1 l-bit value.

Normally, a lo-bit by 1 I-bit divide yields up to a 19-bit result. But in our case, the values are not independent and we know the result fits in an 8-bit number. Figure 6 shows the defuzzification process graphically.

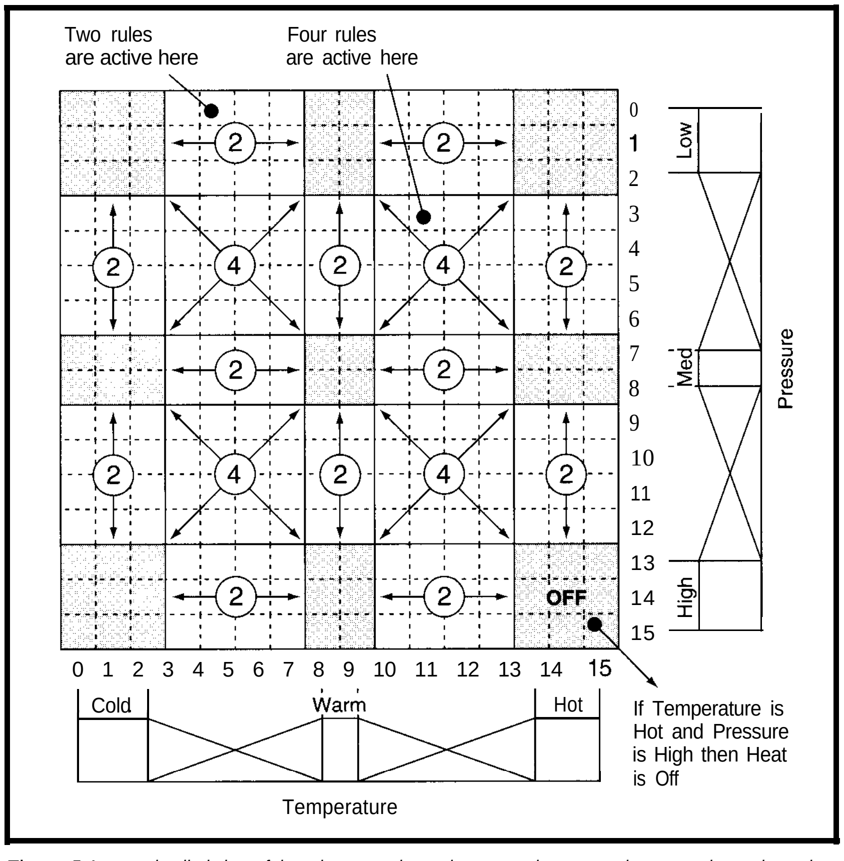
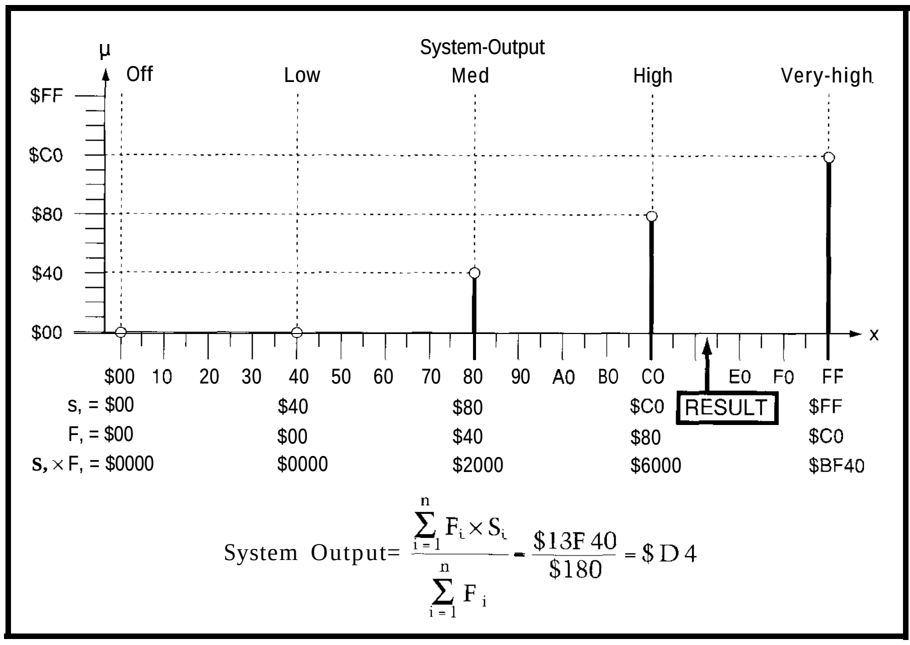

Two rules are active here
Four rules are active here
2
4
OFF
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Low
Med
High
Pressure
Cold
Warm
Hot
Temperature
If Temperature is Hot and Pressure is High then Heat is Off

Figure 5-A more detailed view of the rule space shows the areas where more than one rule can be active at a time. Membership functions for temperature and pressure are shown below and to the right of the rule space.

AN ALTERNATE OFF-LINE APPROACH TO FUZZY LOGIC

When fuzzy logic was first introduced, it was thought to require a lot of processing horsepower. If you choose to use floating-point calculations and complex shapes for membership functions, this is true.

By using simple shapes such as trapezoids and singletons, we greatly simplify the calculations for fuzzification and defuzzification. By using fixed-point calculations in which truth varies between



$$\text{System Output} = \frac{\sum_{i=1}^{n} F_i \times S_i}{\sum_{i=1}^{n} F_i} = \frac{\$13F40}{\$180} = \$D4$$

Figure 6—*This* figure demonstrates the defuzzification process in which three fuzzy outputs are active at the same *time to different degrees. The result* is *the* weighted average of *all active fuzzy outputs.*

$00 and $FF, we eliminate the need for floating point.

Some of the first embedded-control applications for fuzzy logic used the more complex floating-point calculations running on a larger computer or workstation. An output value was calculated for every combination of inputs to derive a control surface. This control surface was then stored in the embedded controller as a large table. During operation of the application, current input values were used to look up the required output in the table.

Although this approach was fast, it tended to require a large memory for the control surface look-up table. It was also difficult to modify this type of system because you had to return to the workstation to make changes and generate a new control-surface table.

CONCLUSION

Fuzzy logic is a powerful and accessible tool for embedded-control applications. It offers a way to work with complex human concepts within a relatively small microcontroller program. This in turn makes it possible to solve problems previously thought to be too difficult for a small microcontroller.

Not surprisingly, many of the first fuzzy-logic applications are traditional control problems in which fuzzy logic replaces another methodology such as PID. The more interesting applications involve new problems in which an embedded controller was previously unable to solve the problem using traditional digital techniques. ❑

Jim Sibigtroth is a system design engineer working on advanced microcontrollers for Motorola. Prior to his work on fuzzy logic, Jim was the systems project leader for the MC68HC11 and wrote the M68HC11 Reference Manual. Jim's other *book,* Understanding Small Microcontrollers *(ISBN* O-23-089129-0), *introduces working engineers to microcontrollers and assembly language programming. He may be reached at jims@seasick.sps.mot.com.*

SOFTWARE

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

I R S

404 Very Useful
405 Moderately Useful
406 Not Useful

If you need I/O...

Add these numbers **up:**

80C552 a '51 Compatible Micro
40 Bits of Digital I/O
8 Channels of 10 Bit A/D
3 Serial Ports (RS-232 or 422/485)
2 Pulse Width Modulation Outputs
6 Capture/Compare Inputs
1 Real Time Clock
64K bytes Static RAM
1+ UVPROM Socket
512 bytes of Serial EEPROM
1 Watchdog
1 Power Fail Interrupt
1 On-Board Power Regulation

It adds **up to real I/O power!**

That's our popular **552SBC**, priced at just $299 in single quantities. Not enough I/O? There is an expansion bus, too! Too much I/O? We'll create a version just for your needs, and pass the savings on to you! Development is easy, using our Development Kit: The 552SBC-50 Development board with ROM Monitor, and an 8051 C compiler for just $449.

New Versions of the 8031SBC

Our popular 8031SBC can now be shipped with your favorite 8051 family processor. Models include **80C51** FA, **DS80C320, 80C550, 80C652, 80C154, 80C851** and more. Call for pricing today!

Truly Low-cost In-Circuit Emulator

The DryICE Plus is a low-cost alternative to conventional ICE products. Load, single step, interrogate, disasm, execute to breakpoint. Only $448 with a pod. For the 8051 family, including Philips and Siemens derivatives. Call for brochure!

Call for your custom product needs. Quick Response.

HTE
HiTech Equipment Corp.
9400 Activity Road
San Diego, CA 92126
(Fax: (619) 530-1458)
Since 1983
(619) 566-1892
VISA MasterCard
Internetmail: info@hte.com
Internet ftp: ftp.hte.com

#120

FEATURE ARTICLE

Robert Schreiber

Levitating a Beach Ball Using Fuzzy Logic

Wanting a hot trade show demo, Microchip takes up Tom Cantrell's PID-pong challenge. Not only do they get a beach ball hovering near the top of a large plastic tube, they do it all with fuzzy logic.

When we took on the task of coming up with a project for a recent trade show, we were inspired by the PID-Pong demo described by Tom Cantrell's "Silicon Update" (INK 42, 50). Tom's demo used a PID algorithm to set the fan speed. In turn, the fan controlled the height of a ping-pong ball in a vertical tube.

However, ping-pong did not fit the trade show's "Beach Party" theme. So, we upped the ante, replacing the ping-pong ball with a beach ball.

DEMO DESCRIPTION

The trade show demo we came up with is shown in Figure 1. A control panel prompts the user to enter the desired beach ball height on the 16-key keypad. The keypad input echoes on the LCD module and the user is prompted for confirmation.

On confirmation of user input, the control panel initiates a ranging cycle to calculate the current height of the beach ball. The desired height and current height are continually displayed on the LCD module. From the current height, the control panel calculates both the velocity and the delta height (i.e., difference in desired height from current height).

This information, along with the desired height, is transmitted to the

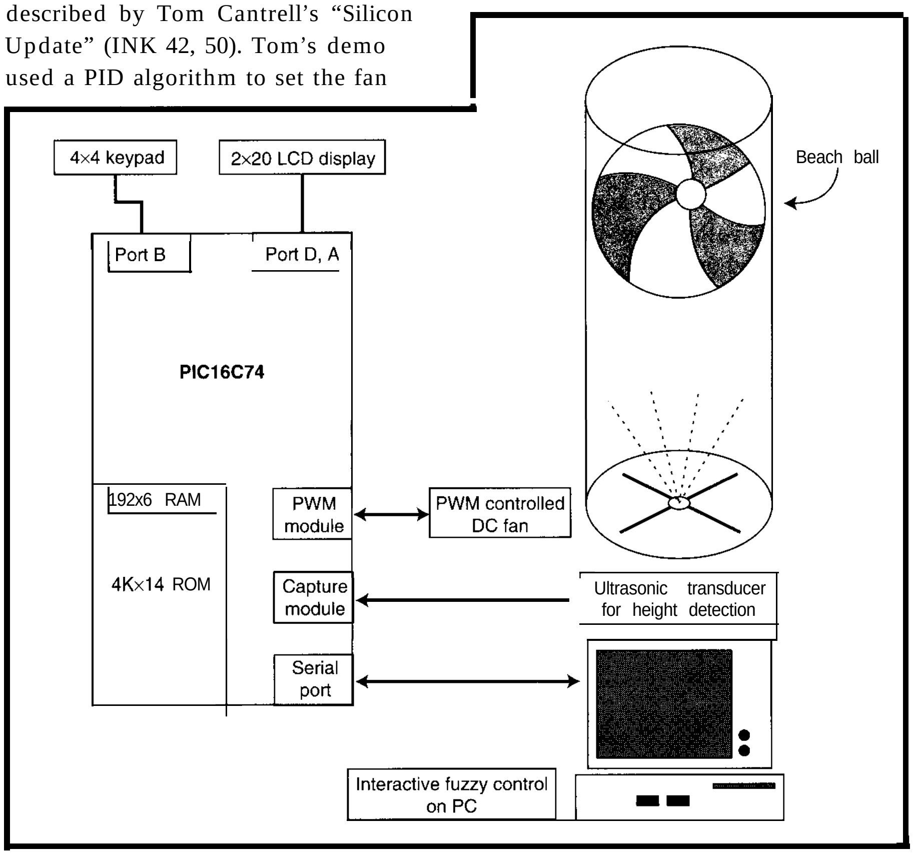


Figure 1—*The trade show demo consists of a control panel and demo cabinet. The demo cabinet contains the DC fan, transducer, power supplies, and the 6" c/ear tube. The control pane/ houses the PIC16C74 microcontroller, which provides the "brains" for all interfaces—PWM DC fan control, ultrasonic ranging (timer capture), keypad decoding, LCD control, and the RS-232 communication to the PC.*

Input variables			Output variable
Current Height	Delta Height	Velocity	Duty Cycle
very lo	neg big	neg big	very slo
lo	neg small	neg med	slo
medium	zero	neg small	medium slo
hi	pos small	zero	medium
very hi	pos big	pos small	medium fast
		pos med	fast
		pos big	very fast

Table I--To *describe the system adequately, a sufficient number of* variables *and terms describing the system must be defined.*

PC via an RS-232 link. The fuzzy logic algorithm, running on the PC, calculates the appropriate duty cycle of the DC fan and transmits this information to the control panel. This emulates a real-world environment in which system-level debugging can be done on the PC in real-time. The control panel controls the duty cycle of the DC fan with this input.

This ranging process continues indefinitely until interrupted by the user. The noticeable differences this project has from the PID-pong project, other than the obvious physical ones, are in the control algorithm and the microcontroller.

The control panel houses an ultrasonic ranging module and the microcontroller. The microcontroller handles all of the peripheral interfaces including the keypad, the LCD display, the ultrasonic ranging module, and the RS-232 serial link.

We wanted a microcontroller that could handle the data throughput and all of these peripherals with little or no external components. The best choice for handling all these functions turned out to be Microchip's PIC16C74.

The PIC 16C 74 contained more than enough on-chip program and data memory. Furthermore, the interrupt capabilities, I/O pins, PWM module, capture and compare modules, timer modules, serial communications interface (SCI), and A/D converter make it a perfect fit for the application. In addition, the on-chip, pulse-width-modulation (PWM) module allows a single-component (FET) interface for the DC fan control.

The ranging module interfaces directly to the microcontroller. The only external component required is a pull-up resistor on the ECHO line because it is an open-collector output. Also, we replaced the gain resistor (R1) for the receiver on the ultrasonic ranging board with a 20-kΩ potentiometer. This enables us to adjust the gain during debugging to reduce reflections inside the tube.

The other major difference from the PID-pong project is the control algorithm. Not only did we have a much larger project than the ping-pong ball, we had a six-week time constraint. This gave us a month and a half to conceive the project and build it to aesthetically pleasing, trade-show standards.

It was enough of a task getting the hardware assembled in the short time frame, but with a PID control algorithm, the project seemed impossible. So, out of desperation, we thought we would put fuzzy logic to the test. We wanted to see if fuzzy logic would deliver on its promises of accurate control and shorter development time. The development tool we used for fuzzy logic control was Inform Software's fuzzyTECH-MP.

Because the hardware development consumed virtually all of the six-week schedule, there was little time left to develop the control algorithm. We didn't really know how the beach ball would behave in the tube or even if we could reasonably control it.

	Shell Value		Code Value	
Variable	min	max	min	max
Current Height	0	120	0	255
Delta Height	-50	50	0	255
Velocity	-5	5	0	255
Duty Cycle	0	255	0	255

Table 2—*The code value is passed to the fuzzy-logic algorithm and is converted to a she// value within fuzzy logic. The shell value is converted back to a code value when fhe fuzzy-logic algorithm outputs if.*

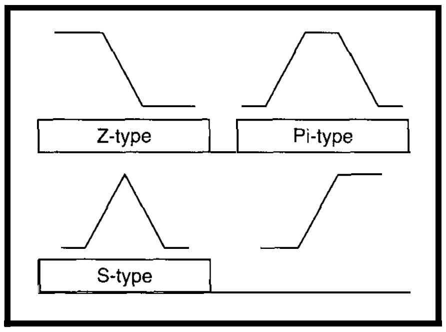
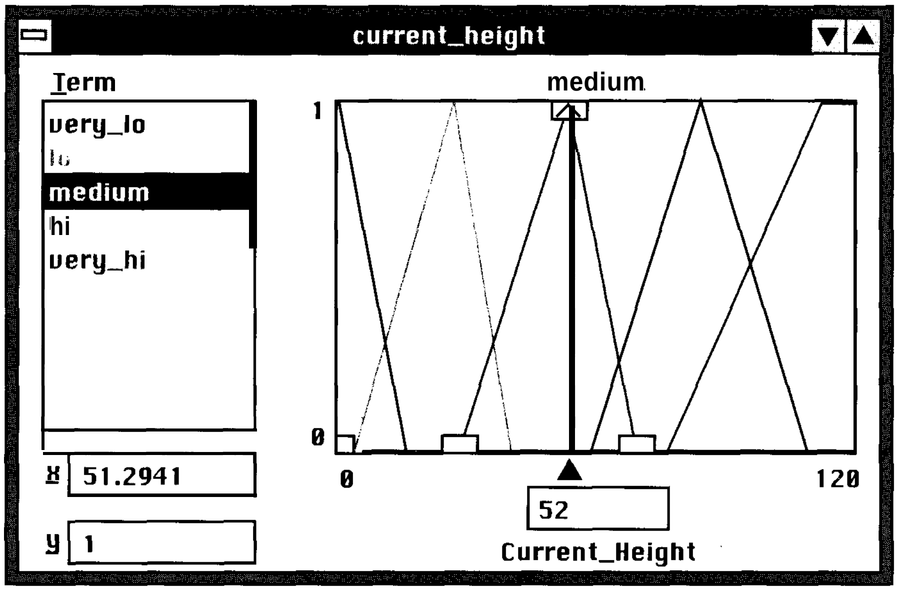


Figure 2—*The standard membership function can be mathematically represented as piecewise linear functions with up to four defining points.*



x 51.2941

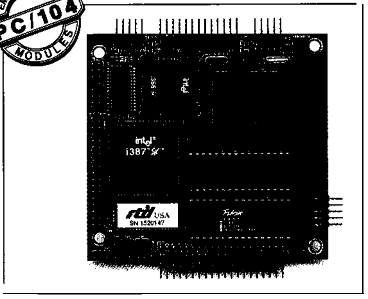
y 1

52

Photo 1—*The* term "medium" for the variable Current *Height* is a Lambda-type membership function *centered around 52. When the beach ball has a value of 52 (or 26"), the degree of membership for fhe beach ball is 1.0 medium. The degree of membership decreases for medium as fhe beach ball moves in either direction from 52.*

REDUCE THE STACK!
Use fully integrated PC/104 CPU and DAS modules from

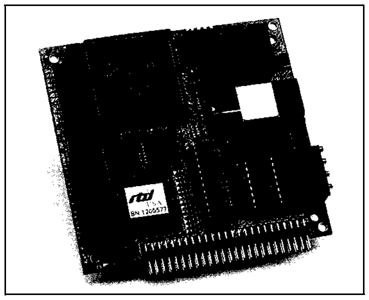
Module size: 90 x 96 x 15mm



PC/104 Compliant PC-AT SBC
CMi386SX-1: $578 /100 pieces
2MB DRAM & SSD software included

5 PC/XT/AT Single Board cpuModules™:
- 486SLC, 386SX, F8680, V41 & VG230 DOS CPUs
- 80387SX math coprocessor socket on-board
- 512KB, 1 MB, 2MB or 4MB DRAM installed
- Two 32-pin SSD sockets & support software
- IDE, floppy & CGA controllers
- RS-232/422/485 serial ports
- Bidirectional parallel, keyboard & speaker ports
- Keypad scanning & PCMCIA support
- Power management & single +5V supply

7 utilityModules™:
- Super VGA controller & I/O modules
- PCMCIA carriers for Types I, II & III cards



PC/104 Compliant 200 kHz Analog I/O Module
DM5408-2: $498 /100 pieces

17 DAS dataModules®:
- 12 & 14-bit A/D conversion up to 200 kHz
- Gap free, high speed sampling under Windows™ & DOS
- Programmable scan, burst & multiburst
- Pre, post & about triggers
- 1 K channel-gain scan memory with skip bit
- 1024 sample A/D buffer
- 12-bit analog outputs
- Bit programmable digital I/O with Advanced Digital Interrupt modes
- Incremental encoder interfaces
- 4-20 mA current loop source
- opto-22 compatibility
- Low power & single +5V power supply

For technical *specifications and data skeets on PC/104, ISA bus and Eurocard products, call*
RTD USA Technical **FaxBack: 1 (814)** 235-1260
RTD USA BBS: 1 (814) 234-9427

&Real Time Devices USA
200 Innovation Boulevard • P.O. Box 906
State College, PA 16804-0906 USA
Tel: 1 (814) 234-8087 • Fax: 1 (814) 234-5218

RTD Europa RTD Scandinavia
Fax: (36) 1 212-0260 Fax: (356) 0 346.4539

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

#121

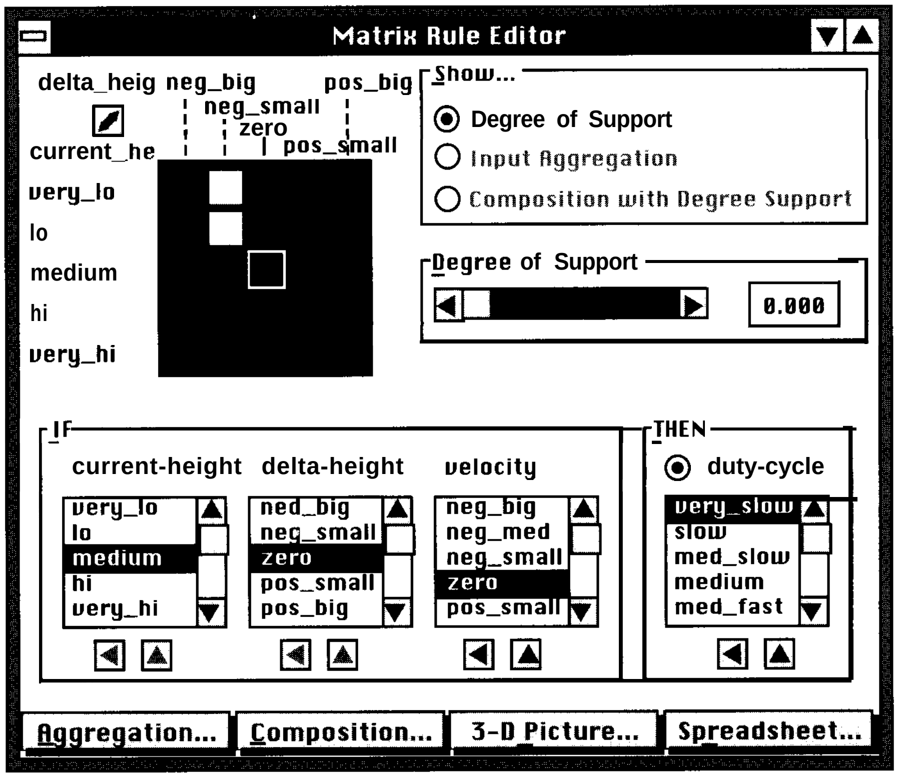


Photo 2—*The Fuzzy Associative Map (FAM) shows the degree of support for each of the rules. For the rule in* this *example, the degree of support is 0,* which indicates a *totally* implausible *rule.*

Finally, five weeks into it, we had the hardware built enough for a manual test. The test was crude, but it did show that control of the beach ball was possible. We at least learned that the algorithm would be able to control the beach ball to within a couple of inches of the desired height.

FUZZY DESIGN

Next, we turned our attention to the fuzzy-logic control algorithm.

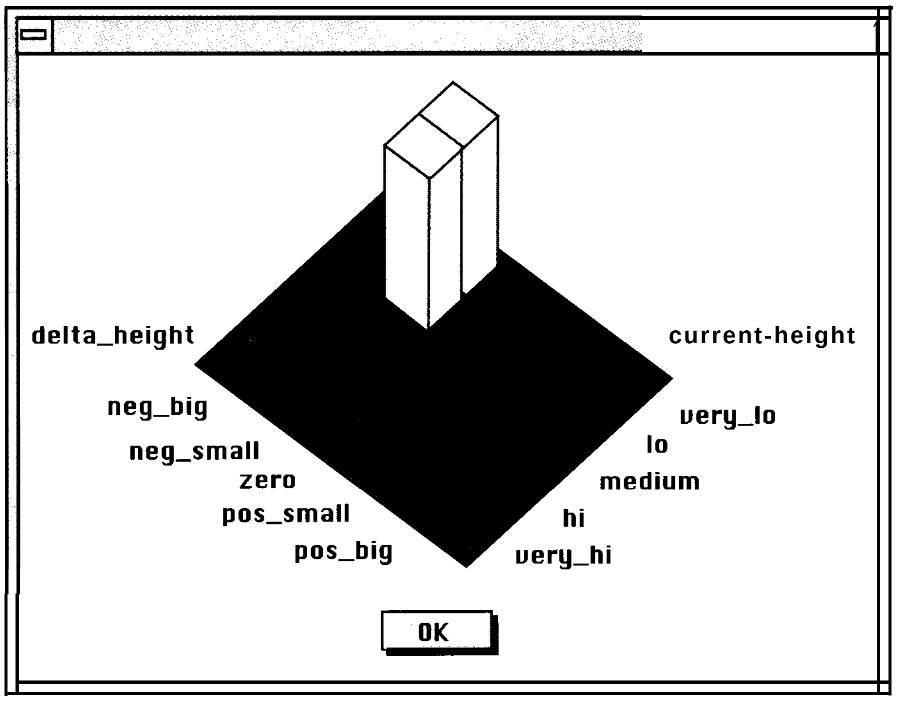
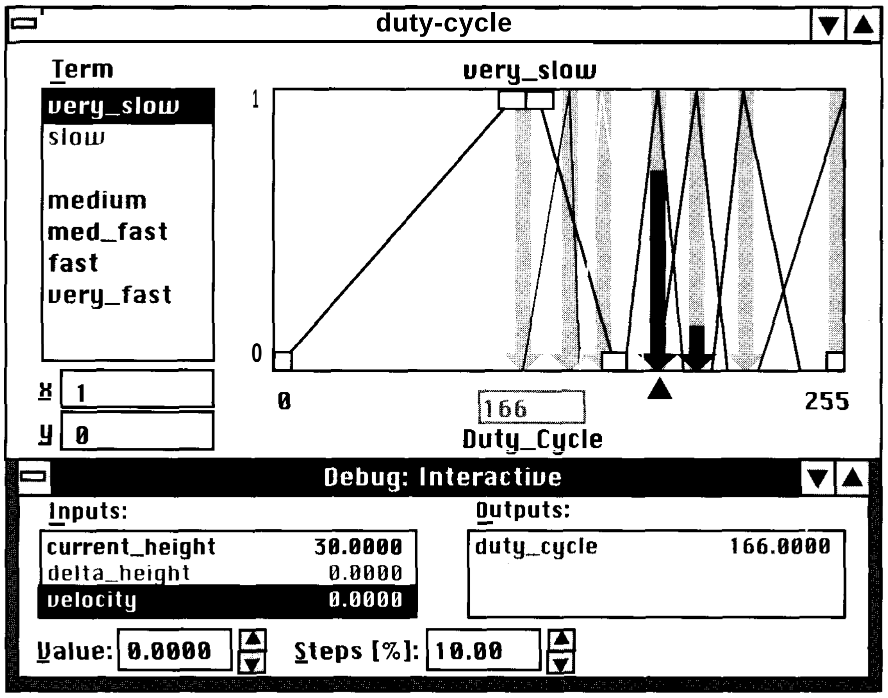


Photo 3—*The* rule *listed in Photo 2 can be represented as a 3D picture.*



```
current_height      30.0000      duty_cycle      166.0000
delta_height         0.0000
velocity             0.0000
```

Photo 4—*The* crisp value is *calculated by an inference weighted mean of the term-membership maxima. That is, the degree of membership for the term medium (long black arrow) is 0.7 and the degree of membership for the term medium fast (short black arrow) is 0.1. The resulting crisp output value is 166.*

Basically, fuzzy logic first translates the crisp inputs from the sensors into a linguistic description. It then evaluates the control strategy contained in fuzzy-logic rules and translates the result back into a crisp value.

Of course, the first step in a fuzzy-logic control design is system definition. This is relatively straightforward for this project. The only possible sources of inputs to the fuzzy-logic control algorithm are the ultrasonic transducer, the user, and the DC fan.

The key is deciding which of these inputs are significant and which aren't. To do this, we put ourselves in the place of the beach ball. We formed a list of critical questions, and for each, we defined a corresponding variable:

- Where am I? → Current Height
- How far am I from where I want to be? → Delta Height
- How fast am I getting there? → Velocity
- What external force will get me there? → Duty Cycle

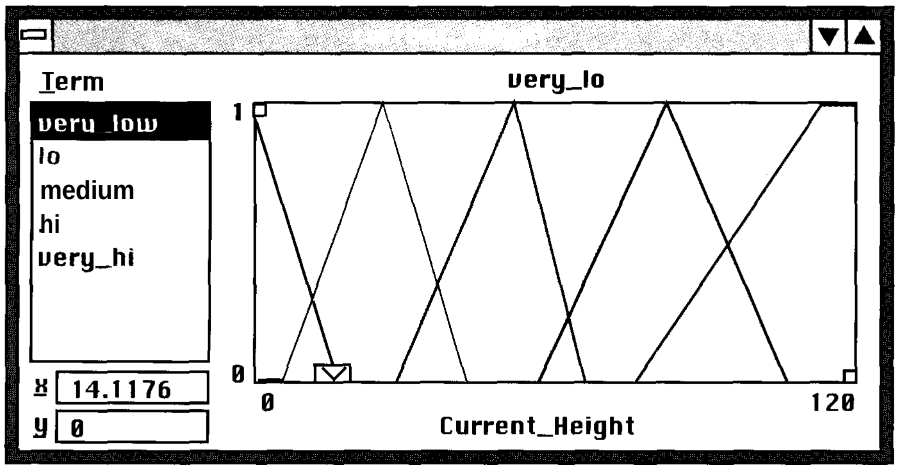


Photo 5—*Once the system-level debugging completes, the final input and output variables are graphically represented. These representations are included in Photos 5–8. Here, although the current height variable contains five terms, we now recognize that three terms would probably have been sufficient. The five terms are fairly symmetrical across the range.*

FREE
Data Acquisition Catalog

1995 **PC and VME data acquisition catalog from the inventors of plug-in data acquisition. Featuring new low-cost A/D boards optimized for Windows, DSP Data Acquisition, and the latest Windows software. Plus, informative technical tips and application notes. Call for your free copy**

1-800-648-6589

ADAC

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

#122

In fuzzy-logic control, the linguistic system definition becomes the control algorithm. And, although defining the variables is the starting point, it isn't good enough to say, "I have velocity." Instead, you need to know to what degree you have velocity.

Determining the extent of a variable is accomplished by defining terms that more fully describe it. The combination of variables and terms gives a linguistic description of what is happening to the system. From this, a variable can be described as having a "positive small velocity" or a "positive big velocity" rather than just a "velocity."

There is no fixed rule on how many terms you need to define a variable. Typically, three to five terms are defined, but more or less may be needed depending on the control algorithm. Table 1 lists the four variables used for the trade-show demo and their associated terms. In retrospect, we probably could have reduced Current Height to three terms and Velocity to five terms.

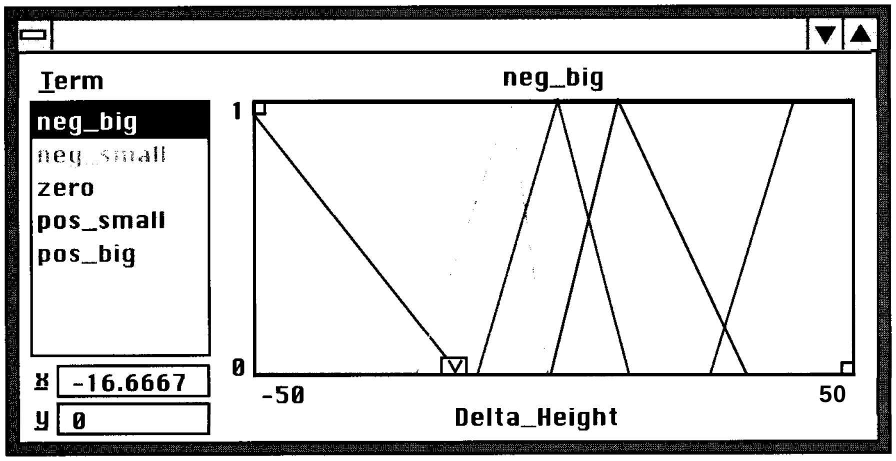


Photo 6—*The delta height* variable contains five terms: neg big, neg small, zero, pos small, andpos *big. The middle terms bunch together around zero.*

Once the linguistic variables are defined, we start defining data types and values. For this application, we defined data types as 8-bit integers and then specified the shell and code values for each variable. The code value is the crisp number that is used in the digital domain and is used when the code is generated. The shell value is the equivalent number used in the fuzzy domain.

For example, you can define the shell value for Duty Cycle to be a minimum of 0 percent and a maximum of 100. Within the fuzzy-logic development tool, Duty Cycle therefore takes on a value between 0 and 100, inclusive.

Similarly, although the code value is limited by the data type, it can take on any or all of the digital range. That is, if the shell value is 0 to 100, the

PC FuzzPong

David Rees-Thomas

For me, fuzzy logic turned Tom Cantrell's PID-Pong into FuzzPong, a fuzzy-logic teaching tool.

The hardware setup is pretty much as Tom Cantrell described in his article (INK 42, 50), except that I used a 12-V centrifugal blower instead of a muffin fan. The duty cycle of a pulse-width modulated (PWM) waveform applied to the gate of a power MOSFET determines blower speed. My ultrasonic rangefinder is an old Polaroid demo kit (unmodified) giving 5 samples per second and a minimum range of about 9".

I do the fuzzy calculations on a PC, so I'm able to add a real-time graphics interface to show fuzzy logic in action. The PC screen (Photo I) depicts the outlines of the input and output membership

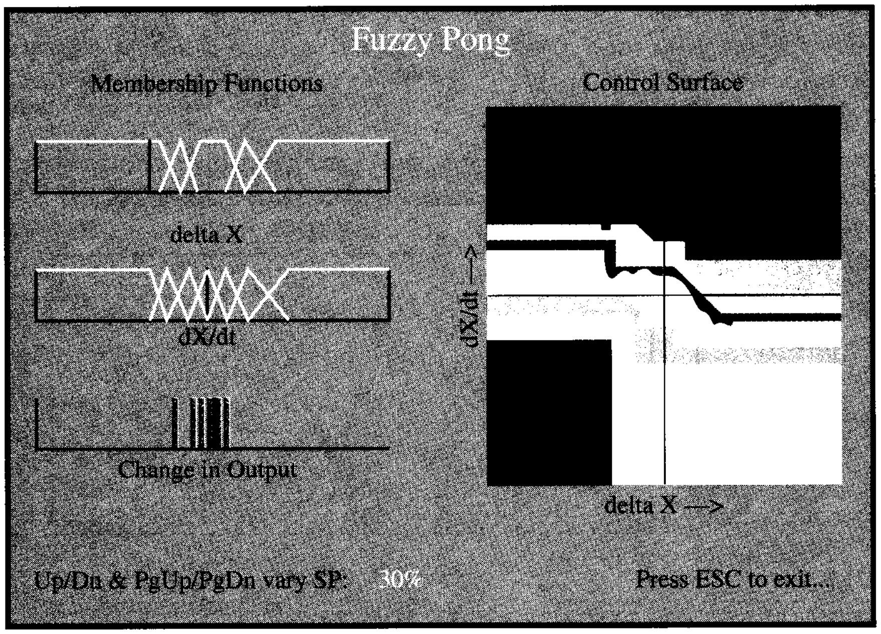


Photo *I-Part of FuzzPong's success as a teaching tool lies* in its graphics display. The control surface shows the *controller output for all possible values of the two input variables. Red indicates areas of large positive change in blower speed, while* blue depicts regions of *large* negative change. The *white region indicates little or no change in blower speed.*

Listing I--The *FTL* (Fuzzy *Technology Language) code ended up compiling down to 0.7 KB of PIC code and used 29 bytes of data memory.*

```
PROJECT {
  NAME = B_BALL.FTL;
  SHELL = MP;
  COMMENT {
} /* COMMENT */
  SHELLOPTIONS {
    ONLINE_REFRESHTIME = 55;
    ONLINE_TIMEOUTCOUNT = 0:
    ONLINE_CODE = OFF;
    TRACE-BUFFER = (OFF, PAR(10000));
    BSUM_AGGREGATION = OFF;
    PUBLIC_10 = ON:
    FAST_CMBF = ON;
    FAST_COA = OFF;
    SCALE_MBF = OFF;
    FILE-CODE = OFF:
    BTYPE = 8_BIT;
  } /* SHELLOPTIONS */
  MODEL {
    VARIABLE_SECTION {
      LVAR {
        NAME     = current-height;
        BASEVAR = Current-Height;
        LVRANGE = MIN(0.000000), MAX(120.000000),
                  MINDEF(0), MAXDEF(255),
                  DEFAULT_OUTPUT(120.000000);
        RESOLUTION = XGRID(0.000000), YGRID(1.000000),
                     SHOWGRID (ON), SNAPTOGRID(ON);
```

(continued)

code values can be 0 to 100. However, to get full resolution, we defined the code values as 0 to 255. The code and shell values are shown in Table 2. Note that for the height and velocity variables, the shell values are scaled by two (e.g., a Current Height with a crisp value of 60 corresponds to 30").

Next, we defined the membership functions that further describe the variables. FuzzyTECH-MP, the fuzzy-logic development tool we used, creates membership functions automatically. Although this gives a good starting point, the membership functions still need to be fine-tuned during debugging. In this application, we used only the linear-shaped functions (Pi, Z, S, and Lambda types) as shown in Figure 2.

FUZZIFICATION

Once the variables are specified, it's time to define the interfaces between the input variables. These interfaces contain the fuzzification procedures, which also need to be defined. For code efficiency, the

functions (MF) and a map of the control surface generated by the current rule base. The instantaneous values of delta_X [distance from setpoint), dX/dt, and change in controller output appear as moving vertical bars.

An MC68HC11 E9 microcontroller does the low-end measurement and control work, communicating with the PC serially at 9600 bps. Input Captures monitor two signals on the rangefinder logic board to give a 16-bit value proportional to the height of the ball. This value is transmitted to the PC as four ASCII characters.

The control value returned by the PC is 8-bit binary and represents a change in the duty cycle of the PWM waveform. A toggle switch selects fuzzy or manual control. In manual mode, a 2-kΩ pot, connected to one of the 'HC1 l's ADC inputs, sets the PWM duty cycle.

FuzzPong is written in Turbo C and takes advantage of that com-

Listing I—*This code fuzzifies crisp inputs, performs max-min composition, and defuzzifies weighted output membership functions to yield crisp output value.*

```
/* F_CONTROL—fuzzy computation on crisp inputs T, Tdot */
unsigned int f_control(unsigned int T, unsigned int Tdot)
{
  int i, j:
  float f_T[9], f_Tdot[9];            /* input membership value */
  float f_out[9];                     /* output singleton MFs */
  unsigned int output:                /* crisp output vale */
  for (j = 0: j < N_T; j++)           /* fuzzification-m_T[] */
    f_T[j] = fuzz (T, m_T[j]);        /* and m_Tdot are input */
                                      /* membership functions */
  for (i = 0: i < N_Tdot; i++)        /* defined in file */
    f_Tdot[i] = fuzz (Tdot, m_Tdot[i]):   /* FUZZYSET.DAT */
  infer(rule, f_Tdot, f_T , f_out0; /* MAX-MIN composition */

  output = defuzz(f_out, m_OUT, N_OUT); /* defuzzification */
  return (output);


/* FUZZ-returns membersh p of input in a fuzzy set */
float fuzz(unsigned int  nput. unsigned int fm[4])

  if ((input >= fm[1]) && (input <= fm[2]))  /* fm[] is MF */
    return (1.0);                            /* definition */
  else if ((input > fm[0]) && (input < fm[1]))
    return ((float)(input fm[0])/(float)(fm[1]    fm[0]));
  else if ((input > fm[2]) && (input < fm[3]))
    return  ((float)(fm[3] - input)/(float)(fm[3] fm[2]));
  else return (0.0);
```

(continued)

computation of fuzzification is carried out at runtime.

In this project, the type of fuzzification used is a membership-function computation. This choice is largely due to the code-space efficiency and accuracy of this method. Once fuzzification has taken place, the algorithm is performed in the fuzzy world according to the rule base.

FUZZY RULE BASE

Next, we are ready for fuzzy inference. The entire fuzzy inference is contained within the rule blocks of a system. For example, if the beach ball is near the top of the tube and we commanded it to be near the bottom of the tube, the rule that describes the situation would be:

```
IF Current Height = very hi
   AND Delta Height = neg big
THEN Duty Cycle = slow
```

Rule definition continues until we have adequately described the system. Note that the IF part of the fuzzy

Listing I-confirmed

```
TERM {
  TERMNAME = very_lo;
  POINTS = (0.000000, 1.000000),
           (14.117647, 0.000000),
           (120.000000, 0.000000);
  SHAPE = LINEAR;
  COLOR = RED (255), GREEN (0), BLUE (0)
}
TERM {
  TERMNAME = lo;
  POINTS = (0.000000, 0.000000),
           (5.176471, 0.000000),
           (24.941176, 1.000000),
           (40.941176, 0.000000),
           (120.000000, 0.000000);
  SHAPE = LINEAR;
  COLOR = RED (0), GREEN (255), BLUE (0)

TERM {
  TERMNAME = medium:
  POINTS = (0.000000, 0.000000),
           (27.294118, 0.000000),
           (51.294118, 1.000000),
           (66.352941, 0.000000),
           (120.000000, 0.000000);
  SHAPE = LINEAR:
  COLOR = RED (0), GREEN (0), BLUE (255)

TERM {
```

(continued)

Listing *I-continued*

```
/* INFER—max-min composition on crisp inputs T, Tdot */
void infer(int rule[][9], float f_Tdot[], float fT[],
           float f_out[])
{
  int i, j, k;
  float cons[9][9];             /* weights of rule o/ps */

  for (i = 0: i < N_Tdot; i++)  /* compute min for each */
    for (j = 0: j < N_T; j++)   /* combination of inputs */
      cons[i][j] = amin (f_Tdot[i], f_T[j]);

  for (k = 0; k < N_OUT; k++)   /* clear fuzzy o/p array */
    f_out[k] = 0.0:

  for (i = 0; i < N_Tdot; i++) /* compute max for each */
    for (j = 0; j < N_T; j++){  /* output membership fcn */
      k = rule[i][j];
      if (f_out[k] < cons[i][j])  /* giving fuzzy weight */
        f_out[k] = cons[i][j];    /* for each output MF */
    }
}

/* DEFUZZ- COG defuzzification of singletons MFs */
unsigned int defuzz(float f_out[], unsigned int m_out[],
                    int n)
{
  int i;                        /* f_out[i] is the fuzzy */
  float crisp   = 0;            /* weight computed for */
  float weights = 0;            /* the singleton output */
                                /* MF m_out[i] */
```

(continued)

piler's graphics library. The program includes three main modules: FUZZMAIN, which contains the graphics routines, FUZZCOMM, which handles data to and from the 'HC11, and FUZZMATH, which is the actual fuzzy controller.

In addition to managing the graphics display, FUZZMAIN runs the executive loop, which keeps the whole show going. FUZZCOMM scales height values received from the 'HC11 and computes their rate of change using a three-point, backward-difference formula. It also massages output data prior to transmission to the microcontroller. Here, I experiment with various software filters and smoothing algorithms, with dubious results.

FUZZMATH (see Listing 1) does its fuzzification, inference, and defuzzification straightforwardly. I stuck to trapezoidal or triangular membership functions for the input fuzzy sets and singletons for the outputs. FUZZMATH performs the usual max-min composition to

Listing 1-continued

```
        TERMNAME = hi;
        POINTS = (0.000000, 0.000000),
                 (55.529412, 0.000000),
                 (82.352941, 1.000000),
                 (106.352941, 0.000000),
                 (120.000000, 0.000000);
        SHAPE = LINEAR:
        COLOR = RED (128), GREEN (0), BLUE (0);
      }
      TERM {
        TERMNAME = very-hi;
        POINTS = (0.000000, 0.000000),
                 (73.411765, 0.000000),
                 (113.411765, 1.000000),
                 (120.000000, 1.000000):
        SHAPE = LINEAR:
        COLOR = RED (0), GREEN (128), BLUE (0)

    } /* LVAR */



} /* VARIABLE-SECTION */

   OBJECT_SECTION {
     INTERFACE {
       INPUT = (current-height, FCMBF);
       POS = -213, -137;
       RANGECHECK = ON;
```

(continued)

inference is aggregation and can be AND or OR.

The rules of the rule block can be defined in terms of plausibility. A plausible rule is defined by a 1.O while a totally implausible rule is defined by 0.0. The degree to which a crisp value belongs to a term is known as the *degree of membership.*

For example, the terms *medium* and *hi* for the variable Current Height are defined as a Lambda-type membership function centered around the crisp values 52 (26") and 82 (41"), respectively, as shown in Photo 1.

Therefore, if the beach ball was at 26", the degree of membership is 1.0 for medium and 0.0 for hi. However, as the beach ball rises in height, the degree of membership for the term medium decreases and the degree of membership for hi increases.

The interplay of these linguistic variable terms is controlled by the rule base, which defines not only the relationship between the terms, but also how much each rule is supported. The support of a rule, or plausibility, is

generate fuzzy outputs, combining them to produce a crisp output value by center-of-gravity weighting.

FuzzPong uses membership functions and a rule base, defined in an ASCII text file (Listing II). It's easy to change the number and limits of membership functions or to tweak the rules so you can see the effect of the changes. Even without a real pong system connected, FuzzPong's control surface shows roughly how the controller reacts in each case. (Note: the surface shows controller action only and *not* overall system response!)

HOW WELL DOES IT WORK?

I haven't made any quantitative measurements, but in the absence of external disturbance, FuzzPong can hold the ball within roughly one ball diameter of the setpoint. It recovers nicely if the system is "bumped" by placing a finger across the end of the tube. Both bumping and a change of setpoint show a

Listing *I-continued*

```
  for (i = 0; i < n; i++){
    crisp += f_out[i] * (float) m_out[i];
    weights += f_out[i];      /* compute weighted average */

  return (unsigned int)(crisp/weights);
```

Listing II- *FUZZYSET.DAT includes fuzzy membership functions and rules.*

```
*Input MFs are entered as four hex values A B C D
* where the MF has the generalized trapezoidal shape:
*               B---C
*              /     \
*             /       \
*            A         D
* N_T (number of MF for first inpu  variable):
   5
* Membership function names:
    NL NS ZR PS PL
* Membership function limits
    0x00 0x00 0x60 0x70
    0x60 0x70 0x70 0x7C
    0x70 0x7C 0x88 0x98
    0x88 0x98 0x98 0xB0
    0x98 0xB0 0xFF 0xFF

* N_Tdot (number of MF for second input variable):
   5
```

(continued)

known as the *degree of support* for that rule.

From the list of rules, a Fuzzy Associative Map (FAM) is constructed. As you can see in Photos 2 and 3, the FAM shows the plausibility (degree of support) of each rule.

DEFUZZIFICATION

The interface for the output variables contains the defuzzification procedures. This project, like most control applications, the center-of-maximum (CoM) method is used for defuzzification.

CoM evaluates multiple output term as valid and makes a compromise between them by computing a weighted mean of the term-membership maxima. The example in Photo 4 shows defuzzification of the linguistic variable Duty Cycle using CoM.

The crisp values of the three input variables used in Photo 4 are:

Current Height:	30
Delta Height:	0
Velocity:	0

Listing 1-continued

```
    }
    INTERFACE {
      INPUT = (delta-height, FCMBF);
      POS = -216, -83;
      RANGECHECK = ON:

    INTERFACE {
      OUTPUT = (duty_cycle, COM);
      POS = 158, -79;
      RANGECHECK = ON;

    RULEBLOCK {
      INPUT = current-height, delta-height, velocity;
      OUTPUT = duty-cycle;
      AGGREGATION = (MIN_MAX, PAR (0.000000));
      COMPOSITION = (GAMMA, PAR (0.000000));
      POS = -39, -113;
      RULES {
        IF    current-height = very-lo
          AND delta-height = neg_big
        THEN duty-cycle = slow   WITH 1.000
        IF    current-height = very_lo
          AND delta-height = neg_small
        THEN duty-cycle = med_slow   WITH 1.000;



        IF current-height = very-hi
          AND delta-height = pos_small
```

(continued)

Listing II—continued

```
*Membership function names:
   NL NM NS ZR PS PM PL

* Membership function limits:
   0x00 0x00 0x50 0x70
   0x50 0x70 0x70 0x7C
   0x70 0x7C 0x72 0x98
   0x82 0x98 0x98 0xC0
   0x98 0xC0 0xFF 0xFF

* N_OUT (number of output MFs)
  5
* Singleton output function (0x80 => zero change):
*    NL   NS   ZR   PS   PL
   0x70 0x7C 0x80 0x82 0x88

* Fuzzy rule base (FAM matrix): the consequent of each rule
* is the index of the corresponding output MF,e.g., 2 => ZR
* Tdot T -> NL NS ZR PS PL
*  NL
                4  3  3  3  3
*  NS
                4  4  3  2  2
*  ZR
                3  3   2 1 1
*  PS
                2  1 1 0   0
*  PL
                0  0  0  0  0
```

fairly heavily damped response. FuzzPong also handles a ball wrapped with one turn of electrical tape without significant loss of control.

All in all, the exercise of writing and using FuzzPong has been a great introduction to fuzzy control. ❑

David Rees-Thomas has a B.Sc. in chemistry and math from Queen's University and a diploma in Electronics Technology from Northern College in Kirkland Lake, Ontario. For the last ten years, he has been teaching at the British Columbia Institute of Technology in Burnaby, BC, where he specializes in microcontrollers and data communications. David may be reached at resd2215@bcit.bc.ca.

I R S

407 Very Useful
408 Moderately Useful
409 Not Useful

Listing 1—*continued*

```
           AND velocity = neg_big
           THEN duty-cycle = very-fast with 1.000;
         } /* RULES */
       }
       INTERFACE {
         INPUT = (velocity, FCMBF);
         POS = -211, -29;
         RANGECHECK = ON;
       }
     } /* OBJECT_SECTION */
   } /* MODEL */
} /* PROJECT */
TERMINAL {
    BAUDRATE      = 9600:
    STOPBITS   =   1;
    PROTOCOL      = NO:
    CONNECTION    = PORT1;
    INPUTBUFFER = 4096;
    OUTPUTBUFFER = 1024;
} /* TERMINAL */
```

The crisp value can be calculated using the CoM method with the following equation:

$$C = \frac{\Sigma i\{I \times \max_x(M) \times \arg(\max_x(M))\}}{\Sigma i I}$$
$$= \frac{(0.7 \times 165) + (0.1 \times 178)}{0.7 + 0.1}$$
$$\approx 166$$

where C is the crisp output value, *i* is the linguistic term, *I* is the inference result, and M is the membership function of the linguistic term.

For this example, when the crisp values are fuzzified, the Duty Cycle variable is defined to be mostly medium (≈0.7 degree of membership) and somewhat medium fast (≈0.1 degree of membership). The arguments for the medium and medium-fast term membership maxima are 165 and 178, respectively. When this fuzzy description is defuzzified, the output is the crisp value 166 as is shown in Photo 4.

SHOW TIME

The first time we ran the demo, the beach ball barely lifted off the DC fan. Apparently, we had our Duty Cycle defined too low. So, in real time, we shifted the Duty Cycle terms to the right and watched the beach ball slowly lift off the DC fan. We adjusted the Duty Cycle so that the beach ball reached 30". We played with the Delta Height terms-we bunched neg small, zero, and pos small-and the beach ball stabilized at 30". There was virtually no fluctuation in the height.

Although 30" was a good starting point, we knew that the system was highly nonlinear. So, we began testing the system at extreme levels and moving the beach ball at different rates from one extreme to the other.

From the manual control tests performed earlier, we had a good characterization of how the beach ball would behave in the extreme regions. It turned out that terms for Current Height and Velocity needed almost no adjustment. In fact, the Velocity variable was not even used.

The variable that required the most work was the Duty Cycle. But before the end of the day, the algorithm was working well beyond our expectations. The beach ball could go from resting, with the DC fan off, to the maximum allowable height of 42" in less than 8 s with no overshoot. Operation between the minimum and maximum height was much quicker, and there was no overshoot.

We felt confident that we could sleep well that night. Ironically, it was the last sleep we got for a while. During the night, a cold front moved in. When we tried to run the beach ball demo the next day, it sent the beach ball to the top of the tube every time.

To make a long story short, the problem turned out to be with the ranging module. The receiver gain was set a little too high. The potentiometer was set just below the level of receiving reflections in the tube. The changes in the environment pushed it over the edge. After a minor adjustment to the potentiometer, we were up and running again.

However, this time, once we started the demo again, the beach ball would stop 6" short of the desired height. After thinking about what else we may have missed, the answer hit us like a blast of cold air-literally.

The cold front changed the atmospheric conditions enough so that the DC fan didn't have enough juice to push the ball up to the desired height. This is where Velocity, our one unused term, came into play. We decided to

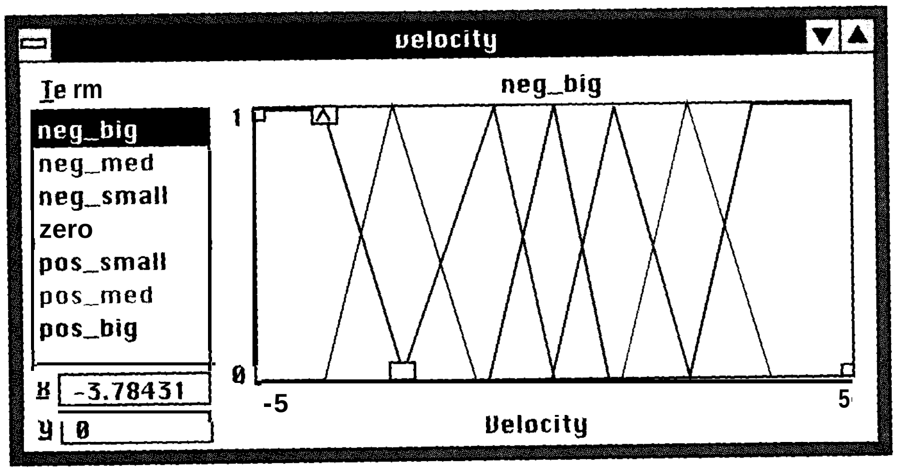


Photo 7—*The* velocity variable contains seven terms: neg *big, neg med, neg small, zero, pos small, pos* med, *and pos* big. *The terms* are *nearly* symmetrical across the range. With *hindsight, we realize that these seven terms could be* reduced *to* five.

add a few rules that used Velocity to nudge the ball into place-you know, as sort of a turbo mode. With this adjustment, the demo worked.

Photos 5, 6, 7, and 8 graphically depict the final state of the linguistic variables. Listing 1 offers an excerpt of the Fuzzy Technology Language (FTL) that we used. (FTL is a vendor and hardware-independent language which defines the fuzzy-logic based system.)

Once we had completed the fuzzy logic algorithm, we ran the assembler to get an estimate of the memory needed to embed it in the PIC16C74. The fuzzy logic algorithm used approximately 0.7 KB of program memory and 41 bytes of data memory. The total code space for the project was 1 KB of program memory and 80 bytes of data memory. Including the fuzzy logic algorithm, we still had well over 50% of the memory resources available on the PIC16C74.

FUZZY CHALLENGE

Our trade show demo was very successful. The positive feedback virtually guaranteed that the demo will surface again at future trade shows. However, now that the public has seen the demo, marketing wants to capitalize on its success by adding enhancements.

Two enhancements are already in the works. The first includes adding manual control to allow a user to challenge the fuzzy logic control. The second entails breaking the serial communication link and embedding the fuzzy logic in the microcontroller.

Finally, if we get crazy enough, we'll remove the tube and run the demo in free air.

So, if you happen to see us at a trade show near you, come put fuzzy logic to the challenge! ▣

Special thanks to John Day, Rodney Duke, and Mort Simmonds for their help with the project.

Robert Schreiber is a senior applications engineer at Microchip Technology. He has more than ten years of embedded systems, hardware, and software design experience. He may be reached at apps@mchp.com.

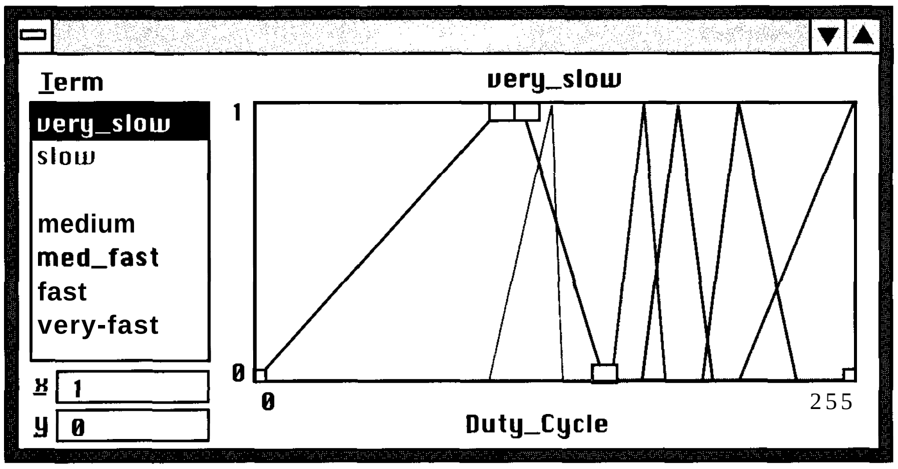


Photo &The *duty cycle variable contains seven terms: very slow, slow, med slow, medium, med fast, fast, and* very *fast. The terms bunch* together around medium.

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

A. ***Compare Avocet Systems with the competition.***

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

Now call the obvious choice!

AVOCET SYSTEMS: INC.

The Best Source for Quality Embedded System Tools

(800) 448-8500

(207) 236-9055 Fax (207) 236-6713

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

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

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

[illegible]

#123

SOURCES

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

Inform Software, Inc.
1840 Oak Ave.
Evanston, IL 60201
(708) 866-1838
Fax: (708) 866-1839

IRS

410 Very Useful
411 Moderately Useful
412 Not Useful

FEATURE ARTICLE

Mike Smith &
Kathy Kim

Being ASSERTive with Your Processor

The Advantage of Software Interrupts

To interrupt or not to interrupt, that continues to be the question. Past articles have explored the advantages of polling and straight-line code. Now Mike takes a look at the benefits of using software interrupts.

Do-While Jones' article on "Interrupt-free Design" (*INK* 43) stressed the need to remember the old computer adage-more haste, less speed. He claimed that introducing interrupts into time-critical code can often be counterproductive.

In this article, I want to take a contrary approach and examine some advantages in deliberately using interrupts. Bear in mind though that the interrupts I'll be covering are in software, while Do-While covers hardware interrupts specifically.

The Motorola's MC68010 is considered in some circles as the first processor to make good use of software interrupts. The 680x0 is a family of processors, which Motorola planned to make code-compatible with future processors. To do this, an actual instruction on a future processor caused an illegal instruction trap on an earlier processor so that the instruction could be emulated in software. Although the code is the same across the processor family, the differences in speed are quite dramatic.

Advanced Micro Devices also chose this approach. The Am29050 processor performs pipelined floating-point and integer operations in a single cycle. The Am29200 microcontroller on the SA-29200 evaluation board [1] performs integer operations in the same single cycle. Floating-point operations, however, are handled via a software trap through the monitor on the evaluation board and take 280 times longer. This approach has one critical advantage. If your requirements change, you can make tremendous cost savings because you can move already developed and tested software to the faster system.

There are other special features on processors using software interrupts. After every arithmetic operation, the programmer should check to see whether results can still be represented in the number of bits available on the processor. On IBM's PowerPC and most other floating-point processors or coprocessors, floating-point instructions are highly pipelined to obtain good performance. Specifically testing for a floating-point overflow condition can be costly. Not only do you have to fetch the compare instruction, you may also have to wait for the flags to pass through the pipeline before you can test them.

Rather than fetching an additional software instruction to test for possible floating-point problems, chip designers take a hardware approach—the floating-point exception. If there is no overflow, then the program continues smoothly. Otherwise, the processor traps and uses the information in the vector table to jump to an error-correcting routine. As Do-While points out, interrupt overhead can be expensive. But, since the traps are not taken frequently, the normal condition is handled more efficiently.

Considering its advantages, it is surprising that the exception idea is not often extended to integer arithmetic. Most processors require that you specifically check for integer overflow.

By contrast, the Intel 80960 processor has integer (signed) and ordinal (unsigned) add and subtract instructions that can be enabled to trap on overflow. The AMD 29k RISC processors operate somewhat similarly. They have specific signed (ADDS and SUBS) and unsigned (ADDU and SUBU) instructions that trap. Other add and subtract instructions can be used when it doesn't matter.

The 29k processors have another interesting class of single-cycle, software-interrupt instructions. These are the ASSERTive instructions (highlighted in the title), which act essentially as a single-cycle compare-and-trap instruction.

Although exceptions from normal integer and floating-point instructions typically occur infrequently, ASSERT instructions can be advantageous even when the exceptions occur as frequently as 30% of the time. The rest of this article examines the advantages of these instructions in a number of DSP situations.

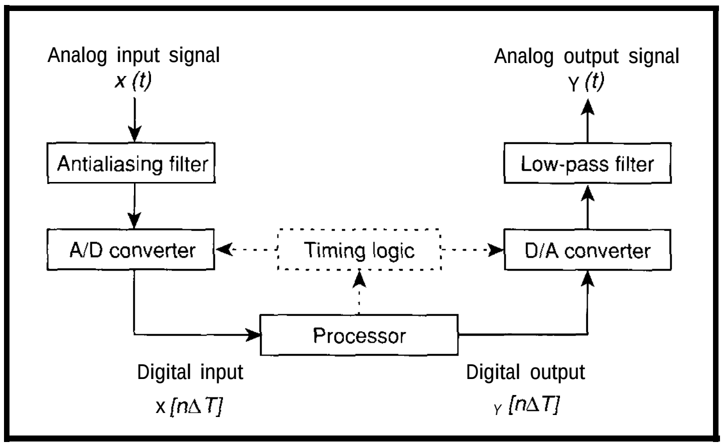


Figure I--The *typical DSP experimental setup for application of a digital finite-impulse-response filter (FIR) digitizes the signal, modifies it in some way, and converts it back to analog.*

FIR FILTERS

A simplistic experimental setup for a digital filter is shown in Figure 1. After conditioning the analog signal to reduce its noise bandwidth, the signal is digitized and then manipulated by the processor. The digital filter's output is converted to an analog signal and finally smoothed to produce the required filtered signal.

Digital filters have the advantage over analog filters in that their components (coefficients) are unaffected by temperature or time. And, if you widen the register and memory data paths, you avoid many of the quantization errors and overflows, which are drawbacks of the digital filter.

To maximize the bandwidth, it is necessary to take advantage of the processor architecture to reduce the number of instructions generating the digital-filter output. The FIR (finite impulse response) filter produces an output $y(n\Delta T)$, which is a weighted (H(i)) average of the last p input signals, $x([n-i]\Delta T)$:

$$y(n\Delta T) = \sum_{i=1}^{p} x([n-i]\Delta T)H(i)$$

Because of the nature of the convolution equation, it is not unusual to find that both the input and output signals can be expressed in 12 bits, yet the internal states of digital filter need 20 bits or more accuracy. These 20 bits correspond to the 12 data bits together with lower guard bits to reduce problems associated with truncation and additional guard bits to reduce the chances of overflow.

An advantage of the FIR filter over the IIR [infinite impulse response) filter is that the output signal only overloads if the input signal becomes distorted for a maximum of p sample periods after the distortion is removed. However, the FIR filter requires a large number of filter taps to obtain any useful filter characteristics (100 or 200 FIR taps compared to 10 IIR taps). Completing the FIR calculations between sampling periods may nearly stretch a processor to its limits. If you want the processor to do anything else, you had better do that efficiently.

Consider this same application with the additional requirement that you need to display the filter output in real time with minimal distortion. Now suppose the input results in the occasional 13-bit numbers going to the 12-bit DAC. With this overflow, a large (13-bit) positive number appears as a large (12-bit) negative number since the display unit truncates the 13-bit number to 12 bits. To avoid such confusion, you must waste cycles continually testing for an overflow.

One approach to getting a better output representation of the signal is to use the saturation-arithmetic concept found on the Motorola DSP-56000 processors. The 56000 accumulator has a large number of bits (56 bits) to avoid overflow during the calculation of the convolution sum.

However, the external memory is only 24 bits wide. If the result in the accumulator is too large to store in memory, then saturation arithmetic means that the largest 24-bit value is saved rather than just the lowest 24 bits of the result. Thus, 0x7FFFFFFF would be stored as 0x7FFFFF (largest positive number) rather than clipped to 0xFFFFFF (a garbage, negative number). For a non-DSP-specific processor, all the output values must be examined with software and truncated when it is necessary to provide saturation arithmetic. Needless to say, this is a time-consuming process.

Listing I-Here's the AMD *29k* code for a software-implemented saturated-arithmetic algorithm.

```
          CPLE boolean, data, maximum  ; if (data>max) goto TOOLARGE
          JMPF boolean, TOOLARGE
          CPGE boolean, data, minimum  ; if (data<min) goto TOOSMALL
                                       ;(following instruction in
                                       ; branch delay slot for speed)
          JMPF boolean, TOOSMALL
          NOP                          ; (unfilled delay slot?)
CONTINUE: Rest of code

TOOLARGE: JMP  CONTINUE                ; data=max; goto CONTINUE
          ADD  data, maximum, 0        ; (in delay slot)

TOOSMALL: JMP  CONTINUE                ; data=min; goto CONTINUE
          ADD  data, minimum, 0        ; (in delay slot)
```

FAST-FOURIER TRANSFORMS

Different, but equivalent problems, can occur when using the fast-Fourier transform (FFT) for digital filtering. Here, the input samples are accumulated until there are 256 or 5 12 values placed in an array. These input values are then discrete Fourier transformed, multiplied by the transform of the filter coefficients, and inverse transformed to give the filtered signal. An industrially related tutorial application of the FFT algorithm is given in reference [2].

The FFT approach has the advantage that it requires a time of the order of $N\log_2 N$ operation compared to the N_p operations of the direct FIR filter. The number of filter taps p does not have to be very large before the apparently more complex FFT approach is the faster one.

The FFT algorithm is simple to implement using floating-point computations. However, there are many processors that do not support fast-float operations for the following reason. The IEEE standard for binary, floating-point arithmetic is to take a number and to break it into fields, which can be stored in a 32-bit register and memory location:

$$(-1)^s \times 1.\text{frac} \times 2^{(\text{bexp}\ 127)}$$

where s represents the sign bit (1 bit], *bexp* the biased exponent (8 bits), and *frac* the fractional part (23 bits).

So, thenumber 10.75 (%1010.11) is stored with $s = 0$, *bexp* = 0x82 (3 + 127), and *frac* = 0x2C0000(%0.01011). The advantage of this approach is that a very wide range of numbers can be represented. The disadvantage is that any mathematical operation requires that all of the fields be manipulated, which takes considerable time unless specialized hardware is available. Software floating operations take orders of magnitude longer than the equivalent integer operations.

BLOCK-FLOATING-POINT FORMAT

Instead of using floating-point operations, you can represent the numbers using the block-floating-point (BFP) number representation, also called *fixed point.* With this method, the value 10.75 (0xA.C (%1010.11)) might be stored as $(0x000AC000 \times 2^{-16})$ in 32 bits. The binary exponent value (2^{-16}) is common (fixed) for all numbers (a block) used in the algorithm.

The advantage of the BFP number representation is that the numbers can be manipulated as if they were integers since the exponent is not explicitly stored and manipulated by software. This gives a tremendous speed improvement over a full software floating-point implementation.

Unfortunately, the range of floating-point numbers is limited. For the above 32-bit example, the range is just below 32,768 to -32,768. A second disadvantage is the precision associated with the BFP number representation. For our example, the smallest change that can be recorded is 1 bit, which corresponds to $\frac{1}{65536}$.

Despite these limitations, 32-bit BFP numbers are extremely useful. A typical DSP application might read a 12-bit, dual-sided ADC and manipulate the results. A 32-bit BFP representation would be fine with a 12-bit value 0xVVV stored as 0x00000VVV if positive and 0xFFFFFVVV if negative. The coefficients, H_i, for the DSP application can be calculated via an off-line floating-point C algorithm and converted to BFP format using $(\text{int})(H_i \times$ 0x10000).

A further problem with BFP occurs when division is required. A division may lose bits from the result that are not recovered by future multiplications. This effect can be reduced by the correct placement of the 12 bits within the 32 bits available. Thus, a BFP representation using the value 0x0000VVV0 provides 4 lower guard bits before any precision is lost. In

Listing **2**—*The code to determine maximum and minimum data values differs little from that for saturation arithmetic.*

```
TOOLARGE:  JMP  CONTINUE               maximum = data; goto CONTINUE
           ADD  maximum, data, 0        (in delay slot)

TOOSMALL:  JMP  CONTINUE               minimum = data; goto CONTINUE
           ADD  minimum, data, 0        (in delay slot)
```

Listing **3**—*This listing demonstrates the ASSERTive approach to saturation arithmetic.*

```
           ASLE 0x80, data, maximum  ; if (data > maximum) trap 0x80
           ASGE 0x81, data, minimum  ; if (data < minimum) trap 0x81
CONTINUE:  Rest of code

 ; TRAP 0x80 Interrupt Service Routine
TOOLARGE:  ADD   data, maximum, 0     ; data = maximum
           IRET                       ; (return from trap)

; TRAP 0x81 Interrupt Service Rou ine
TOOSMALL:  ADD   data, minimum, 0     ; data = minimum
           IRET                       ; (return from trap)
```

	Am29200	Am29245	Am29240	Am29000	Am29050
Format	integer	integer	integer	integer	integer/float
Purpose	controller	controller	controller	processor	processor
Bus	single	single	single	Harvard	Harvard
Instruction Cache	—	4 KB	4 KB		—
Data Cache	—		2 KB		—
Branch Cache	—		—	0.5 KB	1 KB
Integer multiply	software	software	hardware	software	hardware
Floating point	software	software	software	software	hardware

Table 1--The *AMD 29k family of processors and microcontrollers offers various architectures suited for specific applications, but retains code capatibility across the line.*

addition, it provides 16 upper guard bits before multiple additions result in an overflow. "Correct" placement depends on the DSP algorithm being programmed.

Although the FFT algorithm does not explicitly involve any divisions, an equivalent problem occurs when it is implemented on an integer processor. A 256-point FFT algorithm is made up of eight passes. At each pass, it is possible for the output values to grow at a rate of $2\sqrt{2}$ relative to the input.

To avoid overflow problems with the BFP numbers, the input values must be downshifted to compensate for this growth. However, each downshift produces data truncation and its associated inaccuracies. As the data growth is only potential, it is important to determine the data maximum and minimum and only scale when necessary. This determination requires essentially identical code to the saturation arithmetic discussed earlier.

THE DIRECT APPROACH TO SATURATION ARITHMETIC

The saturation-arithmetic operation on non-DSP-specific processors must be handled in software via several compare instructions. The code in Listing 1 is for the AMD 29k RISC processors. To maximize speed on these systems, the data to be checked, together with the maximum and the minimum allowable values, are stored in one of the processor's many registers. Listing 1 includes the code segment check for possible overflow.

When programming a pipelined processor for efficiency, you must take into account the possible RISC processor stalls associated with changes in program flow. We have manipulated the code segment so that the normal flow, which is no overflow, has minimal delay (i.e., no jumps). By doing this, we are able to maximize the performance.

We have also optimized the code on overflow by placing useful instructions in the RISC processor's J M P delay slots. A slightly different coding approach is possible with a superscalar processor such as the PowerPC. This processor has different branching instructions depending on whether you

Listing 4-This test code compares the efficiency of COMPARE and ASSERT with saturation arithmetic on a 29k processor.

```
        .global start
        .equu    LARGE, 0x80                ; (Vector table offset)
        .equ     SMALL, 0x81
        .equ     EXIT-SERVICE, 1            ;(HIF service calls)
        .equ     VECTOR_SERVICE, 290
        .equ     ARRAY-SIZE, 1000

        .set     boolean, gr96              ; (Register declarations)
        .set     outaddress, lr2
        .set     inaddress, lr3
        .set     counter, lr4
        .set     value, lr5
        .set     maximum, lr6
        .set     minimum, lr7
start:                                      ; (Set up the traps)
        CONST    lr2, LARGE                 ; (trap number)
        LADDR    lr3, ISRLARGE
        CONST    gr121, VECTOR_SERVICE
        ASNEQ    0x45, grl, grl             ;(HIF request)

        CONST    lr2, SMALL                   (trap number)
        LADDR    lr3, ISRSMALL
        CONST    gr121, VECTOR_SERVICE
        ASNEQ    0x45, grl, grl               (HIF request)

        CONST    maximum, 5                  ; (Set the limits)
        CONSTN   minimum, -5

 ; FIRST TIMING POINT
        INV                                 ; (Invalidate the caches)
; Normal COMPARE test
        LADDR    inaddress, array           ; inaddress = &array[0]
        LADDR    outaddress, array1         ; outaddress = &array1[0]
        CONST    counter, (ARRAY_SIZE2);    (Set loop counter)

LOOP1:  LOAD     0, 0, value, inaddress       while (counter >1)
                                                value = *inaddress;
        CPGT     boolean, value, maximum        if (value > maximum)
        JMPT     boolean, TOOLARGE                value = maximum;
        CPLT     boolean, value, minimum        if (value < minimum)
        JMPT     boolean, TOOSMALL                value = minimum:
        NOP                                   (Can't fill delay slot
CONTINUE1:
        ADD      inaddress, inaddress, 4        inaddress++;
        STORE    0, 0, value, outaddress        *outaddress = value:
        JMPFDEC  counter, LOOP1                 counter--:
        ADD      outaddress, outaddress, 4      outaddress++;

 ; SECOND TIMING POINT
        JMP      ASSERT
        NOP

TOOLARGE:
        JMP      CONTINUE1                  ; (Fix if too large)
        ADD      value, maximum, 0
TOOSMALL:
        JMP CONTINUE1                       ; (Fix if too small)
        ADD      value, minimum, 0

; ASSERTive approach
ASSERT:
        INV                                 ; (Invalidate the caches)
        LADDR    inaddress, array           ; inaddress = &array[0]
        LADDR    outaddress, array1         ; outaddress = &array1[0]
        CONST    counter, (ARRAY-SIZE 2);   (Set loop counter)
```

(continued)

want to prefetch the branch code or the fall-through code.

The checking required for the FFT scaling operations is identical. But, as you can see in Listing 2, the operation to be performed on failing the comparison is different.

If we assume that there are N data values to check, of which L is too large and S, too small, we then know it takes $(N - L - S) \times 5 + L \times 5 + S \times 7$ instructions for this code segment to execute. Similar results (within 10–20%) can be found with other processors depending on their architecture. The actual time required depends on the number of clock cycles per instruction.

Even within the AMD 29k family of processors, the time varies widely. On the Am29050 floating-point processor, there is a branch target cache where the JMP target instructions can be stored. As a result, the number of clock cycles is equivalent to the number of instructions.

By comparison the Am29200 microcontroller has neither branch nor instruction cache. Thus, there is a processor-stall penalty roughly equivalent in time to two instructions every time the processor has to take a jump and reestablish the instruction pipeline. In this situation, the effective number of instructions becomes $(N - L - S) \times 5 + L \times 9 + S \times 11$.

AN ASSERTIVE APPROACH

On all processors, DSP or otherwise, optimum performance is obtained by taking into account the architecture of the processor. The 29k processor ASSERT instruction is a specialized compare instruction. If the assertion is true, the instruction effectively acts as a NOP. However, if the assertion is false, a trap occurs to a service routine. The trap overhead is fairly small since 29k processors are register oriented.

Listing 3 offers an example of saturation-arithmetic code using the ASSERTive instructions. This routine takes a total time of $(N - L - S) \times 2 + L \times 4 + S \times 4 + (L + S) \times$ trap overhead, which is close to 2% times faster than the original code provided the trap overhead is not too large.

Listing *4-continued*

```
LOOP2: LOAD    0, 0, value, inaddress         while  (counter  >1){
                                                value = *inaddress;
       ASLE    LARGE, value, maximum            if (value > maximum)
                                                  value = maximum:
       ASGE    SMALL, value, minimum            if (value < minimum)
                                                  value = minimum;
       ADD     inaddress, inaddress,4           inaddress++;
       STORE   0, 0, value, outaddress          *outaddress = value:
       JMPFDEC counter, LOOP'2                  counter--:
       ADD     outaddress, outaddress, 4        outaddress++

; THIRD TIMING POINT
       CONST   lr2, 0                         exit(0);
       CONST   gr121, EXIT-SERVICE
       ASNEQ   0x45, grl, grl

ISRLARGE:
       ADD     value, maximum  0               (Fix if too large)
       IRET
ISRSMALL:
       ADD     value, minimum  0               (Fix if too small)
       IRET

array: .rep    (ARRAY_SIZE/10
       .word 0
       .word 4
       .word 6
       .word 4
       .word 0
```

(continued)

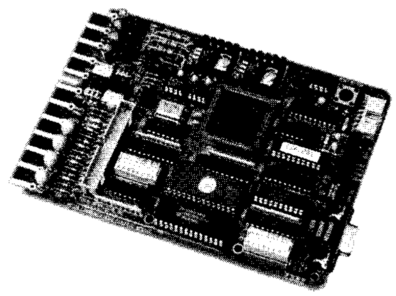
NEW 8051 Family Development System

OEI552 *with **Fuzzy** Logic **Support***

DX126 Control Board
- **Powerful 80C552 Microcontroller**
- **Memory up to 64K ROM/ 32K RAM**
- **AI/DI-DO/PWM Signal Conditioning**
- **Expansion I/O Header**
- **Configurable CODE/XDATA Overlays**

Software Features
- **OEI552 Monitor/Debugger**
- **FL1552 Fuzzy Logic Inferencing Engine**
- **DGU552 Diagnostic Pa&age**
- **Fuzzy Logic Control Template**
- **Numerous Implementation Examples**

Also Available
- **Fuzzy Logic Development Tools**
- **Software Development Tools**
- **Hdw Accessories & Add-on Modules**
- **Sfw Drivers & Control Packages**



This is Not a Toy!

The OEI552 development package is a member of OEI's powerful TRIACS™ family of industrial I/O control modules. OEI offers a full line of industrial control products and services utilizing both conventional and expert system technology.
Call today for more information.

OEI Ortech Engineering Incorporated

17000 El **Camino** Real, Suite **#208**. Houston, TX 77058
Phone: (713) 480-8904 . Fax: (713) 480-8906

#126

MORE HASTE, LESS SPEED?

Obviously, the speed advantage of the ASSERT instruction is optimistic as no code segment ever stands alone. It is also necessary to take into account the effect of any loop overload, the fetching (from memory] of the data value to be examined and its output to the D/A converter, as well as the initial overhead of setting up the trap handlers.

As Do-While Jones states in his article on hardware interrupts, it is important to know when you are ahead of the game! So, we need to decide on which processor architecture you profit from this sort of programming trick and when you are worse off.

One of the reasons that I write a lot about the Advanced Micro Devices RISC processors is that these let me armchair design with a great deal of ease. As Table 1 illustrates, the 29k family of RISC processors has a wide range of system architectures.

However, they also all have an identical instruction set. Instructions such as floating-point operations are supported across the family either directly by hardware or through software emulation, so no recoding is necessary.

I can therefore redesign the chip, check my results on simulators, hassle the guys on the AMD 29k hot line when the numbers don't come out the way I expect (or want), and suggest how it should have been done. All this fun without having to wade through a tremendous learning curve of new instructions and architectural peculiarities for each processor. Armed with this knowledge, I have a jump-start knowing what to look for when I want to discuss the performance of other manufacturer's CISC, RISC, and DSP chips.

We can test the performance of the ASSERTive approach with the code segment in Listing 4, which accesses values from memory as would occur in the FIR and FFT algorithms. The data is checked and manipulated (if necessary) before being stored back to memory. The host interface (HIF) operating system places the start of the interrupt-service routine into the vector table.

Listing 4—*continued*

```
        .word -4
        .word -6
        .word -4
        .word 0
        .word 2
        .endr
array1:
        .block (ARRAY-SIZE * 4); (Output array -- normal approach
array2:
        .block (ARRAY_SIZE * 4); (Output array -- assert approach
```

THE REASON WHY?

The results for the standard and ASSERTive approaches for the various architectures are shown in Table 2. These results provide an interesting insight into the effect of various system features on RISC performance.

There are no conflicts between instruction and data fetches since the Am29000/050 processors have a true Harvard architecture. The Am29200 microcontroller is doubly disadvantaged with two wait states on memory accesses and instruction and the data fetch conflicts. Both the Am29240 and Am29245 microcontrollers have an effective Harvard architecture because of their internal instruction cache. Although slower in their initial memory access of instructions (one wait state), once the instructions are stored in the instruction cache, the effective instruction access time is zero wait states.

Given the Harvard architecture of the Am29050 processor, you might expect that this processor would be at least as fast as the Am29240 microcontroller with its instruction cache. In addition, since both the Am29240 and the Am29245 microcontrollers have instruction cache, their performance should be identical.

However, this is not the case. The difference is associated with the fact that the Am29240 has a two-stage output buffer associated with the data cache. This means that the use of the data bus for a STORE instruction better meshes with the data bus use for a nearby LOAD instruction. This difference gives a data cache and buffer architecture a small 3% speed improvement over a Harvard dual-bus architecture and a whopping 20% improvement over the single-bus architecture.

In practice, the performance improvement depends on how the compiler and programmer organizes the positions of the various LOAD and STORE instructions. It would be interesting to see the effect if the dual-output buffer was added to the Am29-

	Am29200	Am29245	Am29240	Am29000/50
Clock Speed	16 MHz	16 MHz	33 MHz	40 MHz
Memory Access	3 cycles	2 cycles	2 cycles	1 cycle
Cache		Instr.	Instr./Data	Branch
	NO OVERFLOW OCCURS			
COMPARE Cycles	38400	12800	10700	11000
ASSERT Cycles	32000	9600	8000	8000
	10% OVERFLOW OCCURS			
COMPARE Cycles	39500	12800	10700	11000
ASSERT Cycles	35000	10800	9100	9000
	20% OVERFLOW OCCURS			
COMPARE Cycles	40800	12900	10800	11200
ASSERT Cycles	38000	11900	10200	10000
Break-even point	32%	30%	25%	33%

Table 2—*Using* the same *AMD 29k* processors as those *in Table 1, we can compare fhe COMPARE and ASSERTive approaches to saturation arithmetic.*

245 microcontroller, which does not have a data cache.

The presence of the data cache and buffer can also be seen during interrupt handling. With the Am29050 processor, the effective time to execute the ADD and I RET instructions is 10 cycles. On the Am29240 microcontroller (with the data output buffer), the time required varies between 10 and 14 cycles, depending on whether the two-stage data-output buffer needs flushing when the interrupt occurs.

The Am29245 microcontroller, which has no data buffer, requires normally between 11 and 14 cycles, although some configurations of the data sequence can handle interrupts in only 9 cycles. I can't explain the lower number of instruction cycles, which I interpret as being associated with the differences of how the interrupt target addresses are fetched from the instruction and data caches.

For the COMPARE instruction approach, the Am29000/50 processors perform exactly as predicted with two additional cycles for a negative underflow than for a positive overflow. However, the processors with instruction cache require between 0.7 and 3.8 extra cycles, which apparently depends on where the underflow occurs in the data stream. I can offer no reasonable explanation for this other than "just because...."

These results are intended to show the programmer when to use software interrupts rather than the standard program-flow approaches. Regardless of the architecture, if it seems that there is less than a 25% chance of overflow occurring, then it pays to be ASSERTive in your programming style. ❑

The authors would like to thank the University of Calgary, the Natural Sciences and Engineering Research Council of Canada (NSERC), and AMD for providing financial support through operating and scholarship funds. We appreciate the help provided by Gerry, Travis, Scott, and Mike on the AMD 29k hotline and by Dan Mann (AMD Austin).

Mike Smith is a professor in the Electrical and Computer Engineering Department at the University of Calgary. He teaches C, assembly ***language programming, and comparative processor architecture. Kathy Kim is training as a research engineer with Bell Northern Research in Ottawa, Canada. Mike Smith may be reached at*** *smith@enel.ucalgary.ca.*

REFERENCES

[1] M. R. Smith, "Computer "TRAIN"ing," *INK 44* (1994): 30-39.

[2] M. R. Smith, "FFT: fRISCy Fourier transforms?" ***Microprocessors and*** Microsystems, 17:9 (1993): 507–521.

IRS

413 Very Useful
414 Moderately Useful
415 Not Useful

!NEW! FROM CIRCUIT CELLAR
PROJECT FILE, VOL. II

$17.95

SAVE 28% OFF COVER PRICE OF $24.95

All NEW Projects to SPARK Your Imagination!

GPZ8 Audio Sampling System
Wiring Your House for the 21st Century
Multiprocessor Architecture using DSP
AND MUCH MORE!!!

VISA, MasterCard, or International Postal Money Order (U.S. funds drawn on U.S. bank only)

Circuit Cellar Project File

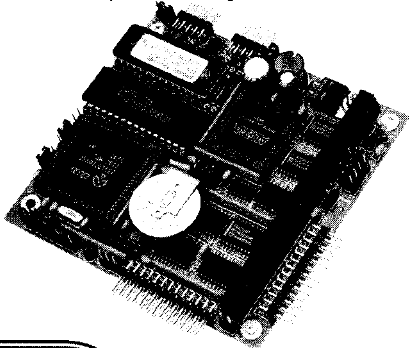
4 Park Street
Vernon, CT 06066
Tel: (203) 875-2199
Fax: (203) 872-2204

'includes domestic delivery. Please add $6 per copy for delivery to Canada & Mexico, add $8 per copy for delivery to other non-U.S. addresses.

Keypads
Enclosures
RS232/RS485
Real-Time Clock
Parallel I/O
Relay Drivers
Battery Backup
ADC/DAC
1M RAM/ROM
LCDs

Low Cost Miniature Controllers

with integrated C-language development system for fast project completion. Call our AutoFAX 916-753-0618 from your FAX. Request catalog #18.



Z WORLD ENGINEERING

1724 Picasso Ave.
Davis, CA 95616
916.757.3737
916.753.5141 FAX

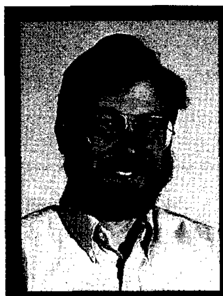
#127

DEPARTMENTS

FIRMWARE FURNAC E

Ed Nisley

Journey to the Protected Land: Smashed Gates & Conforming Code



Enforcing strict levels of access is good for preventing an errant task from trashing sensitive core code, but there are times when such access is necessary. Ed looks at what's involved in granting just that.

no quotes, anecdotes, or malapropos this time around. Got those caffeine heebie-jeebies after flattening a couple of nuisance bugs. It's time to get down to work.

The grand topic this month is controlling low-level access to high-privilege kernel routines. The solution involves call gates, language conventions, synthesized instructions, one new segment, and two major nuisances. Hang on!

CALLING THROUGH GATES

As you saw last month, the '386 passes control to a different privilege level through a call gate. Gates aren't strictly necessary for our code because the FFTS kernel and taskettes all run at most-privileged Level 0. Thus, there is no distinction between kernel and user code, no protection between tasks, and no fettered memory and I/O access. Later on, when we create Level 3 user tasks, these gates will be vital. Until then, they are just a convenient way to organize our code.

Passing through a call gate is a simple matter of executing a FAR CALL with the gate selector in the segment part of the address. The offset part of the CAL L address is unused because the CPU extracts the target procedure's code-segment selector and offset from the gate descriptor fields. A low-privilege task cannot access higher-privilege segments and thus can "see" and execute only those PROCs identified in call gates. Assuming you install sufficient checking and verification in each gate routine, pesky low-level tasks cannot crash the system.

Although gates require selectors, our real-mode tools can't put protected-mode selectors into ordinary FAR CALLs. Because TaskDispatch has no parameters, I could synthesize a FAR CALL instruction with the simple SysCall macro shown last month. Extending this macro to pass parameters on the stack is straightforward (see Listing 1). Note that I've renamed SysCall to CallSys for consistency with other macros we'll meet later on.

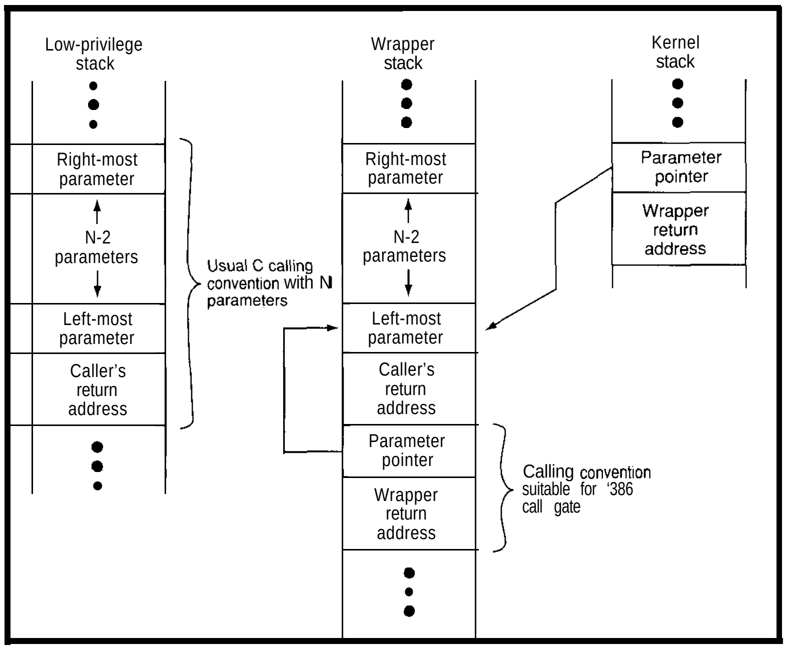


Figure 1—The *FFTS* code passes parameters with the C convention *of right-to-left PUSHes, stacking the left-most argument just above the return address. Because '386 call gates accept only fixed parameter lists, a wrapper routine converts variable argument lists into a single pointer. The CPU copies call gate parameters to a higher-privilege stack, leaving the original list behind on the original caller's stack.*

In another bit of terminological misdirection, the *Intel '386 Programmer's Reference* refers to values on the CPU's stack as *parameters* while the Borland assembler doc describes CALL instructions and macros as having *arguments. I've* tried to keep these terms separate, but probably the best advice is to regard them as synonyms while keeping any subtle differences in mind.

A similar fog engulfs Intel's procedures and Borland's functions. Both refer to the chunk of code executed by a CALL and ignore the Pascalian quibble about returning a value versus just causing side effects. I'll attempt to use C functions and assembly procedures, even if I can't keep a straight face throughout the whole affair.

The FFTS assembly code uses the C-language function-calling convention, meaning that the assembler pushes arguments onto the stack from right to left. CallSys invokes PushArgs, which then calls itself recursively to simulate this convention in the synthesized CALLs. Each invocation of PushArgs cracks its argument list into two pieces. Recursion halts when PushArgs finally encounters an empty list. The recursed levels then unwind with each pushing its left argument and incrementing a counter. Honest, it works!

CallSys then inserts a FAR CALL opcode, a zero-offset address, and the gate's selector into the code segment. Some operating systems place parameters or diagnostic values into the offset, then extract it by reaching back through the gate to the caller's stack. If you need that level of trickery, feel free to go ahead as Intel defines the offset address as "not used," rather than "reserved."

The C-calling convention leaves stack cleanup in the hands of the caller. CallSys observes this require-

Listing 1—Passing *a* fixed list of parameters through a *call gate is simple* enough *because gates are invoked with ordinary FAR CALLs. The CallSys macro pushes its arguments in right-to-left order, then* synthesizes a FAR CALL instruction with the gate's selector as part of the address. The PushArg *macro counts the number of parameters so that CallSys can adjust the stack after the called function returns. All parameters must be 32-bit quantities for this to work correctly.*

```
         MACRO      PushArgs LeftArg,ArgList:REST

         IFNB       <LeftArg>
         PushArgs   ArgList
         PUSH       LeftArg
ArgCount =          ArgCount + 1
         ENDIF

         ENDM


         MACRO      CallSys FnSel:REQ,ArgList:REST

ArgCount =          0
         PushArgs   ArgList

         DB         09Ah             ; CALL LARGE FAR (6-byte imm)
         DD         0                ; 4-byte offset is unused
         DW         FnSel            ; 2-byte selector for call gate

         IF         ArgCount NE 0
         ADD        ESP,4*ArgCount   ; discard stacked parameters
         ENDIF

         ENDM
```

ment by keeping track of the values it pushes on the stack and removing them after the function returns. Although TASM handles this automatically while processing a CALL statement, it seems macros such as PushArgs cannot determine the size of their arguments. PushArgs simply counts the number of arguments it pushes on the stack, CallSys multiplies that number by four to get a byte count and then adds the result to the ESP after the CALL instruction.

Call gates generally reside in the GDT where they are available to all tasks with enough privilege to use them. When you are working with many gates, it makes sense to build a call-gate table and transfer many gates into the GDT with a block move rather than hand-crafting each gate.

Listing 2 shows how I implemented the copy operation. Each of the FFTS kernel's functional areas defines a table in the_protconst segment, which holds all of the kernel's call-gate structures. During the FFTS initialization process, the kernel calls the set-up routine in each area to prepare its own GDT call gates. Remember that the CPU can't use call gates that are not in either the GDT or LDT. The gates in _protconst are useless until they're copied to the GDT.

Procedures accessed by call gates are no different than any other kernel procedures, save that they must end with a FAR RET instruction. The FFTS kernel code can thus invoke them through a call gate using the CallSys macro or directly with a FAR CALL using the CallFar macro. The two macros are identical except that CallFar inserts the target procedure's offset into the CALL instruction.

Up to this point, we have worked entirely in the SMALL memory model with a fixed set of segment-register values. Now that each task has separate code and data segments, the kernel must aim DS, FS, and GS at its own GDT descriptors rather than (mis)use whatever the calling task provides. I'll adjust the code to reflect this as we add more call gates.

The call-gate-descriptor structures include a CopyCount field specifying the number of 32-bit parameters on the caller's stack. When the caller is less privileged than the callee, the CPU switches to a different stack for the duration of the procedure. It automatically copies those parameters onto the new stack before executing the procedure, then copies them back before returning. This prevents the less-privileged caller from finding any trace of the more-privileged callee on its stack. The CopyCount field isn't used for transfers between code on the same privilege level because the CPU doesn't switch stacks.

The TSS defines the three additional stacks required to protect transitions into Levels 0 through 2. A CALL from Level 3 to Level 0 switches into the stack defined by the StackPtr0 field in the TSS (see Listing 4 in

Listing 2—*The code in (a) defines a block of call-gate structures in fhe _protconst segment and copies them to the GDT during the FFTS initialization process. Each gate structure contains fhe code segment and starting address of a FAR procedure along with the number of doubleword (32-bit) parameters if expects to find on the stack. The procedure in (b) copies a block of descriptors info either a GDT or LDT. Accessing a descriptor fable, as always, requires a data alias because the GDT has no descriptor and LDT descriptors do not allow read/write operations. After clearing the target selector's low-order three bits, if is numerically equal to the descriptor's offset in the fable-data segment.*

a)
```
            SEGMENT     _protconst

            LABEL       SysGates BYTE

            DESC_GATE   {OffsetLow=SMALL MemfMakeLinear, \
                        CSSelector=GDT_CODE,             \
                        CopyCount=2,                     \
                        Access=ACC_CALLGATE              \


            DESC_GATE   {OffsetLow=SMALL MemfPeekReal,   \
                        CSSelector=GDT_CODE,             \
                        CopyCount=2,                     \
                        Access=ACC_CALLGATE              \


GATELENGTH  =           $   SysGates
NUMGATES    =           GATELENGTH / 8

            ENDS        _protconst

<<< called during setup >>>
            CALL        MemCopyDescBlock,SYS_MEMORY,GDT_GDT_ALIAS, \
                        GDT_CONST,OFFSET SysGates,NUMGATES
```
b)
```
            PROC        MemCopyDescBlock
            ARG         Selector:DWORD,TableAlias:DWORD, \
                        InSeg:DWORD,pIn:DWORD,NumDesc:DWORD
            USES        ECX,ESI,EDI,DS,ES

            MOV         ES,[TableAlias] ; set up target pointer
            MOV         EDI,[Selector]  ; convert selector to offset
            AND         EDI,0000FFF8h   ; strip LDT and privilege bits

            MOV         DS,[InSeg]      ; set up source pointer
            MOV         ESI,[pIn]

            MOV         ECX,[NumDesc]   ; set up block length
            SHL         ECX,1           ; . . . in DWORDS

            REP  MOVS  [DWORD PTR ES:EDI],[DWORD PTR DS:ESI]

            RET

            ENDP        MemCopyDescBlock
```

INK 54 for details). The operating system must reserve stack areas for each task before allowing entry to the task. Are you getting a glimmer of why FFTS runs at Level 0?

The presence of CopyCount in each call gate raises an interesting question: how do '386 gates handle a procedure with a variable number of arguments? We don't have to look far for an example. Good old StrFormat is a case in point.

POINTERS TO PARAMETERS

I'll spare you the extended rant on RISC versus CISC architectures that should occupy this space. Suffice it to say that call gates work perfectly well for Pascalian procedures with a fixed number of parameters and not at all for C-oid functions like printf() and StrFormat() with an arbitrary number of parameters.

The C calling convention puts the left-most argument on the stack just above the return address. The remaining arguments (if any) follow in left-to-right order in ascending addresses. A wrapper procedure can pass a single pointer to the argument list through a call gate to the kernel routine. Regardless of the number of parameters on the original caller's stack, the gate handles only a single FAR pointer.

The kernel code, running at a more-privileged level, reaches back through the pointer to read the parameter list on the original caller's stack. Figure 1 shows how this works. If a privilege level change occurs, the kernel's stack does *not* have the original caller's parameters because the CPU copies only the parameters specified in the call gate's CopyCount field. The caller's stack is accessible to the kernel as a data area because it is, by definition, at a lower privilege level.

It's worth noting that privilege level changes do not cause a task switch. Except for CS:EIP (and possibly SS:ESP), the CPU registers remain unchanged and the caller's LDT remains in effect. The kernel must load the segment part of the wrapper's pointer into another segment register to ensure that the original caller's stack is accessible since SS points elsewhere.

Listing3 shows StrfFmtFull, the wrapper I put around StrFormat FirstArg is a text macro defined by the assembler that boils down to a displacement from the EBP register. That register has a completely different value in the kernel code, requiring an LEA instruction to compute FirstArg's actual offset address in the stack segment.

The parameters stacked beyond FirstArg are accessed through the pointer and, thus, need not be mentioned explicitly in StrfFmtFull's ARG directive. I defined FirstArg as DWORD:? to indicate that more values may or may not follow even though the question mark is just syntactic sugar. In fact, if the format string does not specify any values from the stack, FirstArg and the corresponding pointer are never used.

Each StrfFmtFull call includes several mandatory arguments in addition to the variable-length list. I decided to pass the fixed arguments unchanged in the hope we wouldn't lose sight of their names. You could, of course, bottle everything up in a single pointer to reduce the amount of stack copying during a privilege transition.

I modified last month's StrFormat to accept a pointer to the variable part of its argument list, renamed it to StrfFmtPtr, changed all the references in the rest of the code, recompiled it, and it works fine. We can now call StrfFmtFull from any task with any number of arguments. It repackages the parameter list and calls the proper kernel routine through a call gate.

However, we're not done with the '386 privilege hardware yet. Recall that a less-privileged task must use call gates to access more-privileged functions because it cannot call them directly. Even though the StrfFmtFull wrapper is an operating-system function, it must execute at the same privilege level as ordinary user tasks.

Not only does the '386 prohibit CALLs to more-privileged routines, it also prohibits CALLs to less-privileged code. Call gates are the only way to transfer between privilege levels. If you want to go slumming in user code, you must get there through a call gate.

In our case, we're wedged. Lowly Level-3 tasks can't call StrfFmtFull if it's at Level 0. Kernel tasks at Level 0 can't call StrfFmtFull if it's at user Level 3. Neither can use a call gate because StrfFmtFull accepts a variable number of parameters. The obvious solution is duplicating

Listing 3—*Call gates transfer a fixed number of parameters defined by their descriptor's CopyCount field and cannot handle functions accepting a variable number of parameters. This wrapper procedure determines the starting address of the variable part of the caller's parameters and passes that pointer as a single parameter. The StrfFmtFull procedure has exactly seven parameters and can be accessed through a call gate, although it's a standard CALL in this code because the target is located in the same source file.*

```
SEGMENT _conform
ASSUME  CS:_conform

PROC    StrfFmtFull FAR
ARG     OutSeg:DWORD,pOut:DWORD,OutSize:DWORD, \
          FmtSeg:DWORD,pFmt:DWORD, \
          FirstArg:DWORD:?
USES    EAX,EBX

LEA     EBX,[FirstArg]  ; EBX = offset of 1st var in SS
MOV     AX,SS           ; set up SS as DWORD
MOVZX   EAX,AX

CALL    StrfFmtPtr,[OutSeg],[pOut],[OutSize], \
          [FmtSeg],[pFmt],EAX,EBX
RET

ENDP    StrfFmtFull

ENDS    _conform
```

StrfFmtFull on each priority level—a kludge if ever I saw one.

CISC got us into this and CISC can get us back out again. It's just another code segment..

CONFORMATION HEARINGS

Ordinary code segments run at the privilege level set by their descriptor's DPL bits. The CPU reads these bits when it branches into the segment and sets the CPL if the transfer is permitted. If the transfer is forbidden, the CPU invokes an error handler. This action is automatic and can't be overridden by the program, which is what makes protected mode so protected.

Something slightly different happens when a CALL instruction enters a code segment that has the Conforming bit set in the descriptor's type field. As long as the target segment's DPL is at least as privileged as the CPL, the CALL occurs without *changing the CPL.* The procedure runs at the same privilege level as the caller, even if the caller is less privileged than the segment's DPL.

As the Intel manual puts it, "Conforming segments are used for programs, such as math libraries and some kinds of exception handlers, which support applications but do not require access to protected system facilities." StrfFmtFull is a classic example of this. It refers only to parameters on the caller's stack, doesn't use any kernel data, and doesn't perform any I/O.

Kernel functions can have their usual direct access to the StrfFmtFull wrapper in a Level 0 conforming-code segment. User tasks at Level 3 can also call it directly without using a call gate. The wrapper runs at Level 0 for kernel callers and Level 3 for user callers, giving both levels access to the same code without special handling or duplication.

The lowercase f in StrfFmtFull reminds us to use a FAR CALL. The FAR procedure declaration in STRING.INC enables assembly-time type checking that can weed out mismatched CALLs and RETs. The code this month isn't entirely consistent with this naming convention for reasons having nothing at all to do with my good intentions. Things should improve next month.

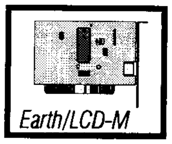
Functions in conforming-code segments may call kernel routines using call gates as usual. There are no restrictions on conforming segments other than the simple and obvious fact that the code must run correctly regardless of the caller's privilege level.

I'm deliberately glossing over the effect of the RPL bits in the various segment selectors and the slightly different rules obeyed by JMP instructions. For now, it's enough that you know conforming segments exist and why they're useful. Check the Official Intel Doc for grubby details before you start your own conformation hearings.

Listing 4 shows the conforming-code segment's selector and segment definitions along with the set-up code that creates the segment descriptor in the GDT. Even though most of our selectors are small integers, I set GDT_CONFORM to 4000h for a specific

---

L.C.D. VGA CONTROLLER



Mono L.C.D.s supported:
640 x 480
640 x 200
320 x 200
480 x 128

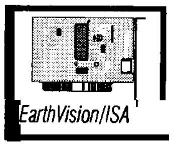
ISA 8 BIT
P.C. COMPATIBLE
WINDOWS DRIVERS
HARDWARE PANNING
RUNS VGA PROGRAMS ON LOW RES LCDs
INCLUDES 10% OFF ANY LCD PANEL DISCOUNT
LOW COST - $149

L.C.D. VGA SCREENS

LOW COST
DISCOUNTED
30%-60%

MONO REFLECTIVE	
VGA 0.4' [IN A CASE!!]	$40
VGA 7"	$50
MONO BACKLIT	
VGA 9.4" 250 MS	$85
VGA 9.4" 150 MS	$99
9" CGA 640 X 200	$19
COLOR BACKLIT	
VGA 9.4" SNGL SCAN	$99
VGA 9.4" DUAL SCAN	CALL
VGA TFT 9.4"	CALL

COLOR L.C.D CONTROLLER



Color L.C.D.s Supported:
Single Scan
Dual Scan
TFT

ISA 16 BIT VGA
WINDOWS ACCELERATOR
512K OR 1 M RAM
CHIPS & TECH 65545
FASTEST LCD CHIP
LOW COST $299

486 P.C. IN A KEYBOARD

33 Mhz 488SX
1-32MB Ram
SuperVGA
2 serial/ 1 Par.
3.5' Floppy
3.5' H.D.

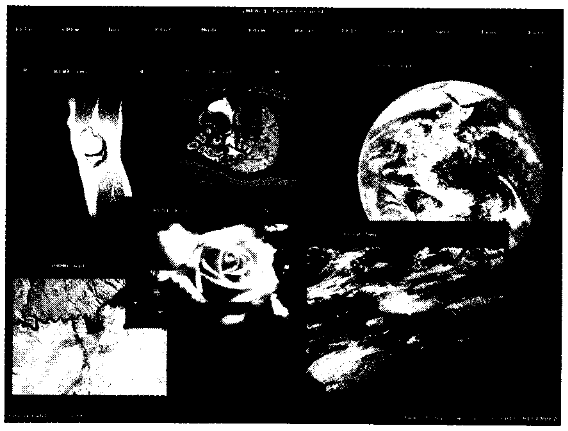
101 KEYBOARD FOOTPRINT
FULLY PC AT COMPATIBLE
ONE 16 BIT ISA SLOT
512K FLASH DRIVE AVAILABLE
ETHERNET DISKLESS FROM $849 RETAIL!

EARTH Computer Technologies
"Innovative Computer Products Since 1984"

Earth Computer Technologies, Inc.
P.O. Box 7069
Laguna Niguel, CA 92607
Phone: (714) 448-9368
Fax: (714) 448-9018

#129

A Serious Imaging Solution



IMPACT Professional is a complete image analysis system that includes a broad range of cross-discipline tools grouped into eight separate processing environments. These modules Include.

- Image Processing System
- 2-D Analysis System for Edge. Shape and Surface Characterization
- Object Counting. Analysis and Sorting System
- Neural Network Graphic Recognition and Classification System
- Image Comparator and Anomaly Detection System
- 3-D Mapping System including Stereo- and Random-dot Projections
- Fractal Analysis System for Point. Line. Edge and Surface Processing
- Color Mapping System which handles Greyscale. Pseudo and 24-bit RGB

SYSTEM REQUIREMENTS: PC/AT or compatible, 386,466 or Pentium, with at least 16 MB of RAM and a hard disk DOS 3.1 or higher. Uses a flat memory model with its own extender and Virtual Memory Manager capable of addressing 4 gigabytes of memory A super VGA video card.

TARDIS Systems
P.O. Box 1251
Los Alamos, NM 87544 U.S.A.

FREE DEMO

Phone: (505) 662-9401
Fax: (505) 662-6780
Technical Support: (505) 662-5623

#130

reason. That choice nearly blew the deadline for this column..

SEGMENT NUMEROLOGY

You should now be quite familiar with the notion of memory segment descriptors in GDTs and LDTs. Each descriptor defines the extent of its memory region through the base and limit fields. The CPU may access only those memory blocks covered by valid descriptors. All other memory is literally out of bounds.

Each descriptor has a corresponding selector that identifies its location in the GDT or LDT. Because the selector has no numeric relation to descriptor's address fields, you cannot compute the selector given a memory address, nor can you derive an address from the selector. All of the familiar real-mode, segment-addressing tricks are invalid in protected mode.

The table ID and RPL bits occupy the low-order three bits of each selector. Thus, if those three bits are zero, the selector is numerically equal to the descriptor's offset from the start of its table. CDT selectors are even multiples of four (0, 8, 16) and LDT selectors are odd multiples of four (4, 12, 20) when the RPL bits are zero.

The FFTS GDT memory selectors tend to be small numbers: GDT_DATA is 18h, GDT_CODE is 30h, and so forth. The LDT selectors in each task are even smaller: LDT_CODE is 0Ch and LDT_DATA is 14h. These are, of course, entirely arbitrary values that could just as easily be near FF00h.

My choice of real-mode tools for this protected-mode series makes segment handling somewhat tricky, as evidenced by the Ca11Far and Ca11Sys macros. In each case, the macro synthesizes a FAR CALL instruction with a specific selector value instead of the normal real-mode segment address corresponding to the target's location. The assembler, linker, and Locate do not change the selector value because it is not part of a standard CALL instruction and does not trigger built-in helper routines.

Ah, but what happens if the protected-mode selector is numerically equal to the real-mode segment address? After all, the selector is an arbitrary number-why not make it useful?

Listing 4—*This code creates the conforming-code segmenf holding the string-formatting functions. The exact order and position of the SEGMENT statements turned out to be critical. They appear after all the other definitions in STARTUP. ASM. Locate copies the code from CONFORM to ROMCONFORM in the binary disk image, and this code simply creates a segment descriptor covering that image. The segment descriptor is numerically equal to the segment's real-mode memory address.*

```
GDT_CONFORM =        (800h SHL 3)            ; 4000h

<<< other SEGMENT statements omitted >>>
            SEGMENT  _conform PARA PUBLIC USE32 'CONFORM'
            ASSUME   CS:_conform
            LABEL    PMConformBase PROC
            ENDS     _conform

            SEGMENT  _romconform PARA PUBLIC USE32 'ROMCONFORM'
            ASSUME   CS:_romconform
            LABEL    ROMConformBase  PROC
            ENDS     _romconform

<<< other set-up code omitted >>>
            MOV      EDI,BASE_GDT+GDT_CONFORM ; conforming code
            MOV      EAX,(OFFSET PMConformLength)-1
            MOV      [GDT_PTR.SegLimit],AX
            MOV      EAX,SEG ROMConformBase    ; segment to linear
            SHL      EAX,4
            ADD      EAX,BASE_LOAD
            MOV      [GDT_PTR.SegBaseLow],AX
            SHR      EAX,16
            MOV      [GDT_PTR.SegBaseMid],AL
            MOV      [GDT_PTR.SegBaseHigh],AH
            MOV      [GDT_PTR.Access],ACC_CONFORM
            MOV      [GDT_PTR.Attributes],ATTR_32BIT
```

Paradigm's Locate (and other similar utilities) makes this step easy enough. Listing 5 shows the Locate configuration file that places the CONFORM class at address 4000:OOO0. The linker combines code in the _Conform segment from all the source files into one block because the segment definition contains the PUBLIC combination keyword.

Ordinary FAR CALLs to procedures in _Conform contain segment address 4000h after Locate finishes fixing up all the segment references. This is exactly the value we'd force with the Ca11Far macro when the protected-mode descriptor for the segment occupies slot 4000h in the GDT. In that case, GDT_CONFORM is, not at all by coincidence, 4000h.

Now, here's the sneaky part. The DUP directive copies the CONFORM segment class into ROMCONFORM. The OUTPUT directive puts that block of code into the disk file. Our boot loader reads the file, pops the whole works into RAM at 1 MB, and passes control to the start-up code that creates a segment covering the _Conform code with a selector of 4000h.

Got it?

This trick works as long as the code doesn't manipulate its segment address. You cannot find the address of a particular procedure in _Conform by shifting the segment address left four bits and adding the procedure's offset. The code is no longer at address 4000:0000 in storage. It's wherever Locate put it in the binary disk image.

You can apply this technique to any segment with a known real-mode address. For example, you might set up a segment descriptor covering the VGA's graphic refresh buffer at A000:0000 with GDT selector A000h. If you have code in the FDB's NVRAM at COOO:OOOO, plunk a descriptor atop it with selector C000h. Sounds pretty easy, doesn't it?

Why not make ***all*** protected-mode segment selectors match their real-mode addresses? I'll leave that as an exercise for you. Hint: consider how you'd get the whole thing running from address 0000:0000. Not all addresses map neatly into PM selectors!

TRIALS & TRIBULATIONS

Nothing is ever simple. I'd planned to present conforming code segments, discuss the changes required to use StrfFmtFull, introduce a new StrfFormat with dynamic memory allocation, and set up a hardware interrupt. Instead, I ran into two peculiar problems.

The CallSys and CallFar macros should work with an arbitrary number of arguments. I generally divide long function calls into several lines with the backslash line continuation character and expected that the macros would follow the same pattern.

It turns out that, for whatever reason, Borland's TASM IDEAL mode does not permit multiple-line macro invocations. The backslash appears as just another argument rather than concatenating successive lines. That confuses the macro and generates a cascade of syntax errors as the assembler attempts to digest the fragments.

I posted a question on Borland's CompuServe forum. Jim Mischel, who literally wrote the book on macros *(Macro Magic with Turbo Assembler)*, replied. It turns out that Microsoft MASM accepts multiline macros or at least it did at one time, while Borland's TASM does not. TASM includes a variety of MASM compatibility features, and Jim suggested several ways to get what I wanted.

The Officially Documented method, the VERSION M510 directive, unleashes a torrent of errors unless it's issued immediately after the PROC directive in each routine and canceled with VERSION T310 and IDEAL directives before the ENDP. I considered replacing all PROC and ENDP directives with macros containing the appropriate hocus-pocus.

Jim pointed out that his code places the (now unofficial) MASM51 directive immediately after the MODEL statement in each source file. I tried it, the files assemble correctly, and it seems to work. I wonder what other effects MASM51 might trigger, though?

Listing 5—*This* Locate configuration file makes the CONFORM segment class *entirely separate from the other FFTS classes. Locate relocates the conforming code to 4000:0000 and then copies the entire class intact to the binary output file immediately after PROTCONST. In protected mode, the code can execute without being copied back to its original location because the real-mode segment equals the protected-mode selector.*

```
hexfile binary offset=00000h size=16 // binary file for boot loade

listfile segments

map 0x00000 to 0x1ffff as rdonly      // startup code normal code
map 0x20000 to 0x2ffff as rdwr        // data segment
map 0x30000 to 0x3ffff as rdwr        // dummy stack segment
map 0x40000 to 0x4ffff as rdwr        // conforming code segment
map 0x50000 to 0xfffff as reserved    // the rest is unused

class    CODE      = 0x0000       // Code (startup + normal)
class    DATA      = 0x2000       // Data
class    STACK     = 0x3000       // dummy stack
class    CONFORM   = 0x4000       // conforming code segment

dup      DATA      ROMDATA        // copy initialized vars to image
dup      FAR_DATA  ROMFAR_DATA    // ditto for far data segments
dup      CONFORM   ROMCONFORM     // conforming code to binary image

order    DATA               \     // RAM organization
         BSS                \
         FAR_DATA           \
         FAR_BSS

order    CODE               \     // ROM organization
         PROTCONST          \
         ROMCONFORM         \
         ROMDATA            \
         ROMFAR_DATA        \
         ENDROMFAR_DATA

output   CODE               \     // Output classes
         PROTCONST          \
         ROMCONFORM         \
         ROMDATA            \
         ROMFAR_DATA        \
         ENDROMFAR_DATA
```

The code this month doesn't use this work-around. I decided to stick with ideal mode's restrictions rather than complicate things further. The monster-size lines hanging off the right margin make the listings awkward enough that I'll insert MASM51 directives later on. For now, this remains an annoying TASM gotcha.

The next problem occurred as I tried to create the conforming code segment at 4000:0000. Depending on the exact placement and order of the _conform and _romconform segment declarations in STARTUP.ASM, either the linker or Locate got befuddled and mislaid the segment. The symptoms ranged from linker errors to baffling run-time protection exceptions.

Of course, I wanted a conforming code segment so we could *finally* get readable, formatted output from error handlers in separate tasks with separate code segments. It does, however, point out the validity of proceeding in small steps rather than giant leaps. I knew pretty much where the problem was, even if I didn't know what caused it. A few hints from the FDB's LEDs, a search over the linker and Locate maps, one smack upside the head, and there it was.

Paradigm's tech support folks offered several suggestions that eventually straightened things out. The ultimate cure was simply rearranging the segment definitions until everything worked. I hate it when code

Acronyms

CPL	Current Privilege Level
DPL	Descriptor Privilege Level
EOI	End Of Interrupt (command)
FDB	Firmware Development Board
FFTS	Firmware Furnace Task Switcher
GDT	Global Descriptor Table
GDTR	GDT Register
IDT	Interrupt Descriptor Table
IF	Interrupt Flag
IOPL	I/O Privilege Level
LDT	Local Descriptor Table
LDTR	LDT Register
NT	Nested Task
P bit	Present Bit (in a PM descriptor)
RF	Resume Flag
RPL	Requestor Privilege Level
TF	Trap Flag
TR	Task Register
TSS	Task State Segment

"just works" without a good reason because it may well fail after a small, seemingly unrelated, change later on. In this case, we just have to live with it because I'm using Locate well beyond the original specifications.

Although you might think a full-bore hardware emulator would make short work of finding such problems, I'm not convinced. The outlandishly clever debuggers we've grown accustomed to can hide myriad assumptions that, in turn, hide the real problem. Better you should know what's going on down at the very lowest levels.

Tradesman, know thy tools!

RELEASE NOTES

The code this month builds a conforming code segment for the string-conversion routines, installs several GDT call gates, and includes various minor tweaks that make it all work correctly. The three taskettes and the task switcher now display running counts on the video screen, one taskette writes to the Graphic LCD panel, and the kernel taskette twiddles the FDB LEDs.

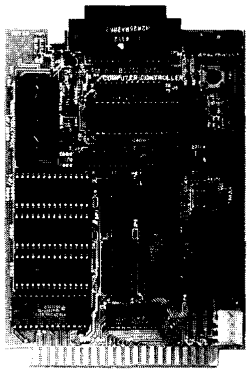
Several folks remarked that they'd like to venture into the Protected Land but lack a Firmware Development Board. Although FFTS writes to the FDB hardware, it can get along just fine with a diskette drive and perhaps a video display. The output-only bits just fall into the bucket. I do recommend building a pair of LED-and-switch widgets to show trace outputs on the parallel ports, though. *INK 3* 1 has the (trivial) schematic.

Next month, we'll replace the error handler with a separate task that produces readable output, activate some hardware interrupts, and see what happens when an interrupt causes a task switch. It's starting to look useful, isn't it?

***Ed Nisley, as Nisley Micro Engineering, makes small computers do amazing things. He's also a member** of **Circuit Cellar INK's engineering** staff. You may reach him at ed.nisley@ circellar.corn or 74065.1363@ compuserve.com.*

416 Very Useful
417 Moderately Useful
418 Not Useful

BCC52
BASIC -52
COMPUTER/CONTROLLER



The BCC52 controller continues to be Micromint's best selling single-board computer. Its cost-effective architecture needs only a power supply and terminal to become a complete development system or single-board solution in an end-use system. The BCC52 is programmable in BASIC-52, (a fast, full floating point interpreted BASIC), or assembly language.

The BCC52 contains five RAM/ROM sockets, an "intelligent" 27641128 EPROM programmer, three 8-bit parallel ports, an auto-baud rate detect serial console port, a serial printer port, and much more.

PROCESSOR
- 80C52 B-bit CMOS processor w/BASIC-52
- Three 16-bit counter/timers
- Six interrupts
- Much more!

INPUT/OUTPUT
- Console RS232 – autobaud detect
- Line printer W-232
- Three 8-bit parallel ports
- EXPANDABLE'

*Compatible with 12 BCC expansion boards

MEMORY
- 48K RAM/ROM, expandable
- Five on-board memory sockets
- Either 8K or 16K EPROM

To Order Call 1-800-635-3355
Tel: (203) 871-6170
Fax: (203) 872-2204

BCC52	Controller board with BASIC-52 and 8K RAM	$1 69.00	Single Qty.
BCC52C	Low-power CMOS version of the BCC52	$199.00	
BCC52I	-40°C to +85°C industrial temperature version	$294.00	
BCC52CX	Low-power CMOS, expanded BCC52 w/32K RAM	$259.00	

CALL FOR OEM PRICING

MICROM I NT, I NC. 4 Park Street, Vernon, CT 06066
in Europe (44)0285-658122 • in Canada: (514)336-9426 • in Australia: (3) 467-7194
Distributor Inquiries Welcome!

#119

The *only* 8051/52 BASIC compiler that is 100 % BASIC 52 Compatible *and* has full floating point, integer, byte & bit variables.

- *Memory mapped variables*
- *In-line assembly language option*
- *Compile time switch to select 805 1/803 1 or 8052/8032 CPUs*
- *Compatible with any RAM or ROM memory mapping*
- *Runs up to 50 times faster than the MCS BASIC-52 interpreter.*
- *includes Binary Technology's SXA51 cross-assembler & hex file manip. util.*
- *Extensive documentation*
- *Tutorial included*
- *Runs on IBM-PC/XT or compa tibile*
- *Compatible with all 805 1 variants*
- ***BXC51 $ 295.***

508-369-9556
FAX 508-369-9549

Binary Technology, Inc.
P.O. Box 541 • Carlisle, MA 0 1741

#131

FROM THE BENCH

Jeff Bachiochi

RF Panic Button Commands Multiple Automotive Functions

Concerned about being stalked in dimly lit parking lots? Try Jeff's RF key-ring transmitter. You'll be able to lock or unlock your vehicle's doors, unlock your trunk, or in case of emergency, sound the horn.

I'm glad November elections are over. There is only one thing I hate worse than political commercials, and that's political yard signs. I can overlook the casual neighborhood sign staked into the ground, but the vacant lots, especially those corner lots, must have vampires buried everywhere.

One night, while watching the tube, one campaign ad caught my wife's attention. A lone female exits a building at night, presumably finishing her second shift. A rather large pair of shoes enters from the shadows, and the sound of footsteps is heard as the woman realizes she is being stalked. A bit distressed, she fumbles her keys, dropping them next to her car's door. The scene changes to Candidate Joe Schmo blasting his opponents' soft line on crime.

As we tune him out, Beverly says to me, "I know that feeling. It's spooky when I get off after midnight. All I can think about is getting to the car, unlocking the door, getting in, and locking it again."

I guess it's time to let her benefit directly from technology. While we don't have an alarm system in our family van, it does have power door locks. This fact makes this project possible.. .well, that and an RF transmitter and receiver pair.

For the remote, I need a small key-ring transmitter with multiple buttons on it (see Photo 1). The button functions include door lock and unlock,

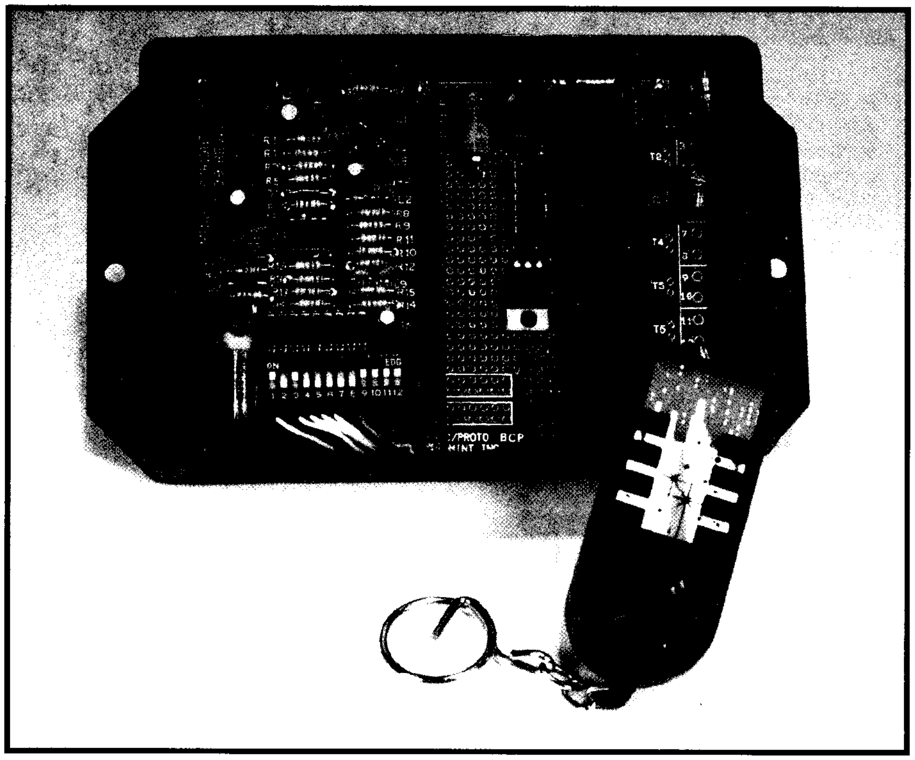


Photo I--The key-ring *transmitter easily* fits into a pocket or purse while the receiver fits *neatly* under the car dashboard.

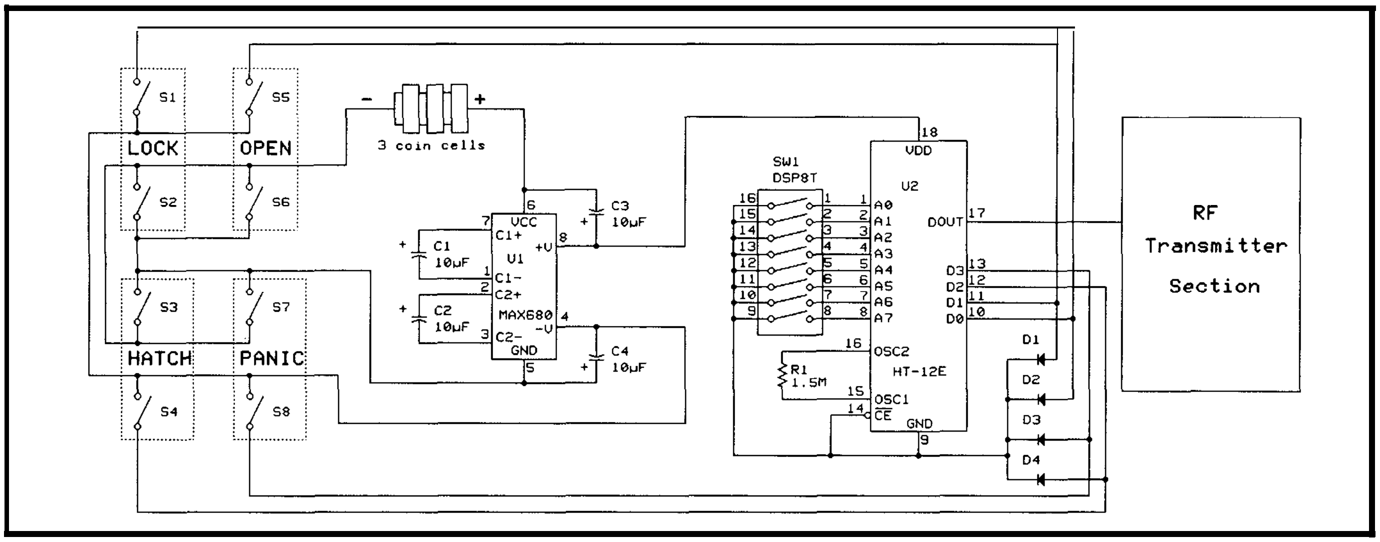


Figure 1—*Two* buttons must be *pressed simultaneously on the transmitter. The first starts the* MAX680 switched-capacitor DC-DC *converter. The second enables a data* stream *from the HT-12E.*

hatch unlock, and panic. The transmission stream contains 4 bits of data and an S-bit address modulated on an RF carrier.

At the receiver, the address and data must be received three times in a row to be acknowledged as "good." After a good reception, a positive-strobe pulse causes the four data bits to be latched. The strobe turns on a small relay with normally open contacts. Three of the data bits control power relays, which are powered only while the strobe closes the relay.

These three data bits momentarily pulse the power to one of three door locks, which is just what the door solenoids are looking for. The fourth data bit latches a fourth power relay enabling a continuous horn blast, a function preferably reserved for the panic or emergency situation. (If you think intermittent blasts are more effective, you could add a slow 1 -Hz oscillator to control the horn.)

TRANSMITTER

Holtek makes convenient encoder and decoder pairs which operate on 3-12 VDC. Bear in mind that 12 V is necessary to get maximum range from the RF modulator. As well, I want the remote to be as tiny as possible, so I use a small lithium cell. Only 2 mA is necessary at 12 V for the encoder-transmitter circuitry. This means, even at 75% efficiency, only about 12 mA will be needed from the coin cell (well within reality].

To provide these requirements, I could use a MAX632—an inductive step-up DC-DC converter-or I could use a MAX680—a switched-capacitor DC-DC converter. The '680 has both positive and negative voltage-doubling outputs. Its outputs are not regulated, but it can convert 3 V to ±6 V (12-V span) with no inductors. The '632 uses a single inductor, but it has the advantage of a regulated 12-V output.

The enclosure probably can be found at a local discount stores. It started life as one of those annoying sound-effects gadgets-a tiny key-chain dongle with eight buttons, each with its own level of auditory pain.

I quickly disabled the noisemaker and found more than the plastic case was useful. The back door of the case exposed three tiny coin batteries. These small cells are a fine source of power. At the opposite end of the case was a nice 1" speaker. (I never seem to be able to find these. It'll find its way into a future project, I'm sure.)

The vacated space is just large enough for the Holtek's HT-12E and an 8-position slide switch. The center section contains a circuit board with a single COB [chip on board) and the interwoven finger-contacts layout for a rubber 8-key button pad. There should be enough room behind this board for the RF transmitter section.

I started by wiring the buttons to be used in pairs. The first of each pair applies power to the DC-DC converter by connecting the battery ground. The second button grounds the transmitter and one of the four data inputs (note the DC-DC converter's +12-V output is always connected to the transmitter). As you can see in Figure 1, the second button's switch contacts connect both a single data input and the HT- 12E's ground when pressed—simplicity using diodes.

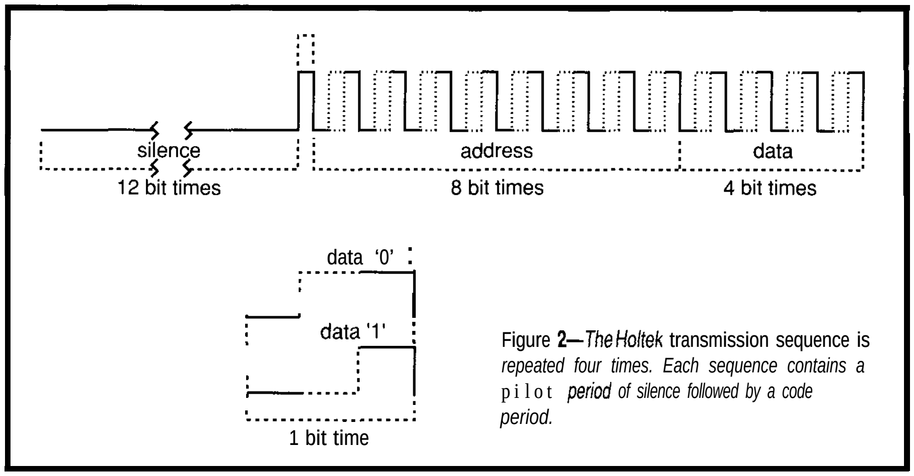


Figure 2—*The Holtek* transmission sequence is *repeated four times. Each sequence contains a* pilot *period of silence followed by a code period.*

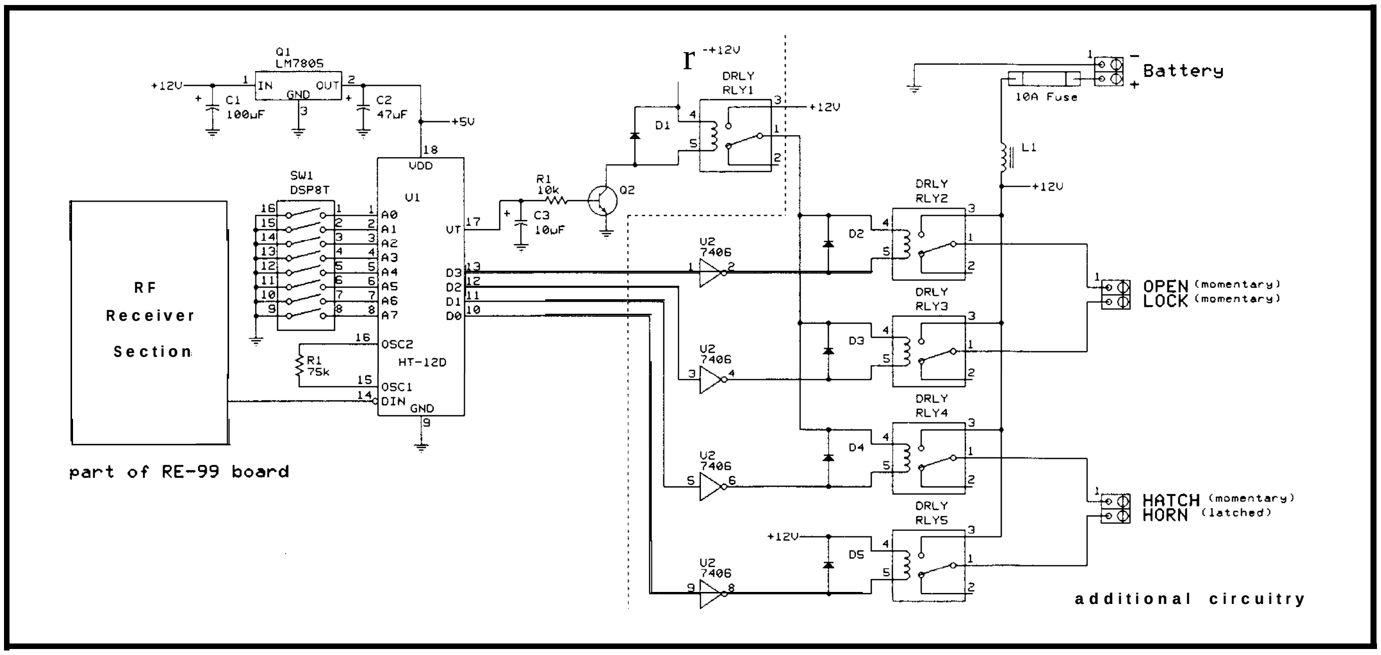


Figure 3-The receiver and decoder provides the strobe and data necessary to control three solenoid controlled door locks and a startling horn blast.

Both buttons (of each pair) must be held to start a transmission. This not only prevents accidental transmissions, but also assures that data remains valid as long as the power is applied to the transmitter.

The HT-12E has twelve inputs. These are broken down into two sections-the address and the data. The address is 8 bits in length or 1 of 256 combinations. The data portion is 4 bits wide and a logic low on any bit indicates a button is being pushed.

An external resister sets the internal oscillator frequency in a range of about 2-6 kHz on the HT-12E. It takes three of these oscillator cycles to send one bit of data. The data format is either low-low-high for a logic 1 data bit or low-high-high for a logic 0 bit.

A single transmission consists of 36 oscillator periods at logic low and 1 period at logic high (the pilot period) plus 12 (8 address and 4 data) 3-period bits (the code period). A minimum of four transmissions are sent each time the transmitter is enabled. Since the enable button is tied low, transmissions are continuous while the buttons are held.

As you can see in Figure 2, carrier is produced when the output of the HT-12E is at logic high. The pilot period clears the air and indicates the duration of a logic 1 time period. The code period carries the actual address and data info. The RF transmitter is a tank circuit connected at one end to Vcc, but held off of ground at the other end by the high-impedance off state of a single transistor. When the HT-12E outputs a logic 1, the drive transistor shorts the tank to ground causing a resonance at about 300 MHz.

RECEIVER

The HT- 12D is the companion decoding device. Its internal oscillator runs about 50 times faster than the transmitter. This enables it to lock onto the logic 1 bit of the pilot period and accurately obtain a reference to compare the following address and data bits. Multiple transmitters, which might be running at different frequencies, work equally well since operation is based on the pilot period of each's transmission.

The demodulated RF reception is fed into the HT-12D, and the address portion of the code period is compared to the eight address inputs on the receiver. If a mismatch occurs on the address bits, the valid transmission (VT) output is lowered and the device looks for more data. If a match occurs on the address bits, which is different from the previously stored 8 address and 4 data bits, the new 12 bits are stored and a counter cleared.

If a match occurs on the address bits and it is the same code period as the previously stored 12 bits, the counter is incremented. Once three identical receptions have occurred in a row, the VT output is raised and the four bits are valid at the HT-12D's data outputs.

ENTRY, CONVENIENCE, SECURITY, AND EMERGENCY

These are the four functions I am implementing at the touch of a button. The first unlocks the doors, the second locks them, the third pops the hatch for easy access when your arms are full of groceries, and the fourth sounds the horn to discourage menacing behavior.

All can be easily controlled by using automotive relays driven by open-collector drivers. The HT-12D is not capable of driving the relays directly since it can only source or sink about a milliamp. The VT output signal is used to control the source voltage for the first three relay drivers, which act as momentary contact closures. The fourth relay is used in a latched mode to be a source of surprise and constant irritation. It can be turned off by simply pressing one of the other functions (see Figure 3).

Next came installing the receiver/ controller in our Caravan. After dismantling the doors and crawling under the dash board, I managed to locate the necessary wiring circuitry. These points must be tapped to give my receiver/controller parallel access to each function. The circuitry is housed in a plastic enclosure small enough to fit behind the center console right beneath the air-heat vents. The four function connections are made with clamp-type electrical taps. For extra protection, ground is attached to the chassis and +V to an unused fuse on the fuse panel.

EXPANDING POSSIBILITIES

The Holtek line has a great number of encoder and decoder chip sets, which range in size from a 2^8-binary address up to a 3^{13}-trinary address. That's ah, urn, a lot of combinations. They transmit from 1 to 8 bits of data in either a momentary or latching configuration. (For more info on using Holtek's chips with RF and IR, see Steve's "Wireless Remote Control of the AVMux" *INK 46.)*

You may wish to experiment with the Holtek chips using any RF or IR transmitters and receivers you have hanging around in one of your junk boxes. An old set of walkie-talkies or maybe some RC equipment can provide you with a working platform. If you have an HCS installed, RF makes a great alternative to the line-of-sight limitations of IR.

Well, I've discovered some valuable information designing and implementing this project. Specifically, I'm glad I don't install car stereos because today's autos are designed for specific equipment with little or no room for extras. The installation of this project was by far the most difficult, but all for a good cause.

Happy birthday, Beverly. ❑

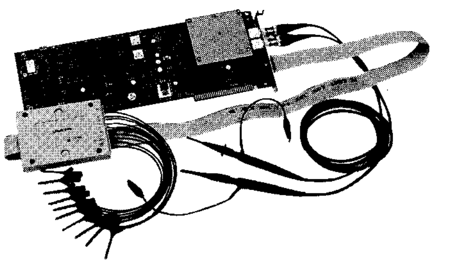
Jeff Bachiochi (pronounced "BAH-key-AH-key") is an electrical engineer on Circuit Cellar INK's engineering ***staff.*** *His background includes product design and manufacturing. He may be reached at jeff.bachiochi@circellar. corn.*

I R S

419 Very Useful
420 Moderately Useful
421 Not Useful

PC-Based Instruments
200 MSa/s DIGITAL OSCILLOSCOPE

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



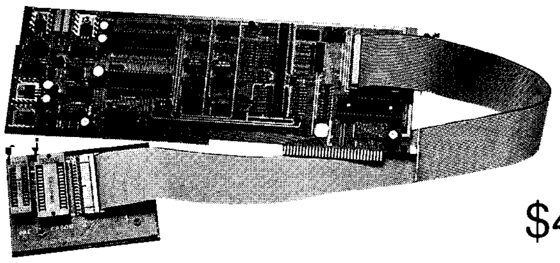
$1799 - DSO-28204 (4K)
$2285 - DSO-28264 (64K)

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

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

Universal Device Programmer

PAL
GAL
EPROM
EEPROM
FLASH
MICRO
PIC
etc..

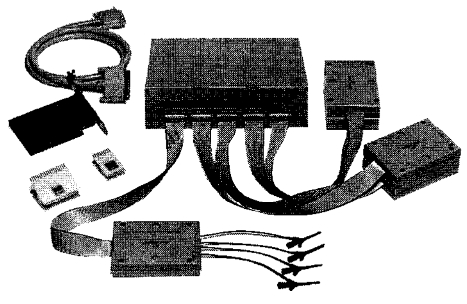


$475

Free software updates on BBS
Powerful menu driven software

400 MHz Logic Analyzer

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



$799 - LA121 00 (100 MHz, 24 Ch)
$1299 - LA32200 (200 MHz, 32 Ch)
$1899 - LA32400 (400 MHz, 32 Ch)
$2750 - LA64400 (400 MHz, 64 Ch)

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

#132

Plan '251 from Outer Space!

Intel's 8xC25 1 SB

SILICON UPDATE

Tom Cantrell

Here's a stamp of approval for Intel's 8xC251SB, a new star upstaging its predecessor, the 8051. After checking out its similarities to the 8051, Tom cuts straight to its differences—alien opcodes, shorter instruction cycles, and register architecture.

No doubt some of you, including my long-suffering editors, are scratching your head over the title. My only excuse is that it wasn't easy coming up with a variation on the UFO (Unidentified Fifty-One) theme of my recent article "UFO Alert" in INK 54.

For those of you who don't know, this title is a takeoff on *Plan* 9 From Outer Space, a circa '50s Hollywood epic widely recognized (justly, I opine) as one of the worst movies ever made.

I saw it on late-night TV a few years back, and it was so bad I can't even remember what Plan 9 was or why it was better than Plans 1-8. The movie seemed to rely mainly on an aging (not gracefully either) Bela Lugosi, who was shuffling to and fro with a sickly grimace. Indeed, I think poor old Bela actually died during the filming, so the geniuses behind this turkey pressed on with a stand-in. Arguably, propping old Bela in the corner would have worked just as well.

I figure about now the normally mild-mannered Intel PR folks have my name called up on their screens, with fingers hovering over the Delete key. So, I should quickly say that their Plan '251 is a heck of a lot better than the ill-starred Plan 9.

PLUG-REPLACEMENT

Like old Bela, the aging 8051 was facing an ignominious end at the hands of more modern and agile competitors. Intel, like everyone else, has periodically tweaked the '5 1 by adding a few snazzy peripheral features and more memory. Unfortunately, this approach has worn thin and by now seems little more than the silicon equivalent of putting a cape on the nearest stagehand and shoving him in front of the camera.

Needing a new star, not just a stand-in, Intel has come up with the 8xC251SB (see Photo 1).

Though a completely new design internally, the '25 1 SB has a familiar face. Taking a look at the pinout (Figure 1), any '5 1 old-timer feels right at home with this familiar 40-pinner. In fact, the pinout is exactly the same as the '51FX derivative, which itself is plug compatible with the original '5 1 (the difference between the two is the multiplexing of additional timer-counter functions on port 1).

Given the duplicate pinout and the fact Intel invented the '51, it's not surprising that the new chip goes to great lengths to handle existing '5 1 software. Though, "compatibility" is a widely abused term, the '25 1 SB is the real thing (i.e., it can actually execute '51 binaries). At the same time, it offers a plethora of new features demanded by finicky customers. More on how the '251SB pulls off this tricky balancing act later.

Not to say that sticking with the '51pinout doesn't result in a gotcha or two. For instance, though the new architecture defines a true NMI (i.e., one that can't be disabled by errant

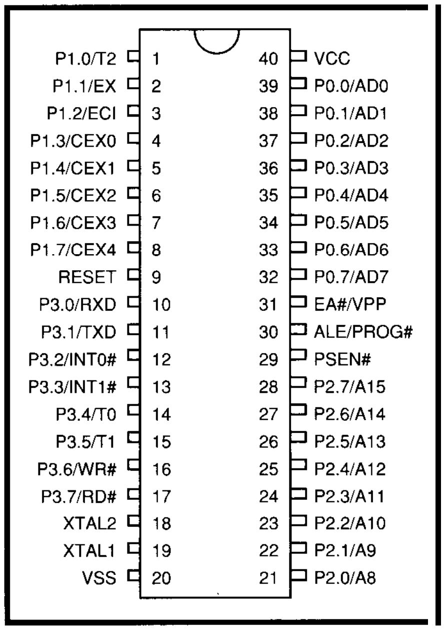


Figure 1—*Other than the multiplexing of additional timer-counter functions on port 1, the '251SB pinout is exactly the same as the original '51.*

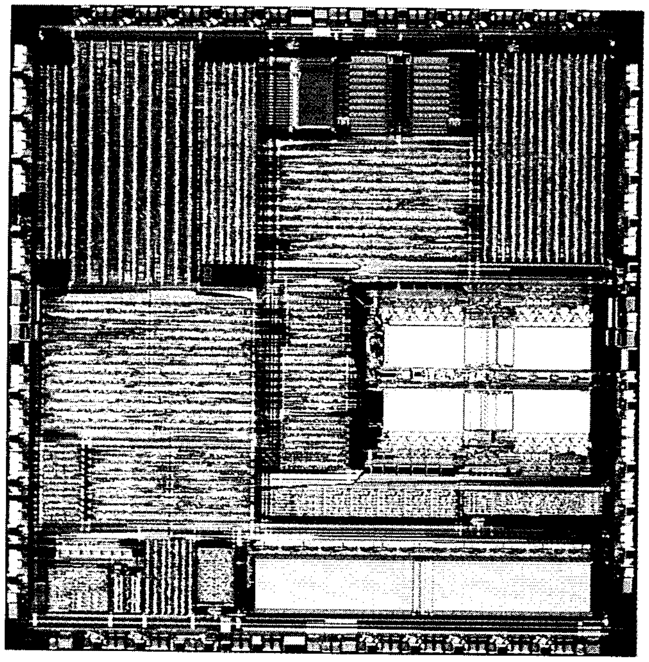


Photo 1—The *Intel '251SB* is the latest in a recent string of UFO (Unidentified *Fifty-One)* sightings.

software), there's no place to put it. You'll have to wait for a future variant of the '251SB for a dedicated NMI pin.

Of course, the only chip that is truly plug-compatible with the one you're using is the one you're using. Since the '25 1 SB is much faster than the '51, be on the lookout for timing loops and other real-time routines that need tuning.

INVASION OF THE LOGIC SNATCHERS

Taking a look inside the '25 1 SB (Figure 2), you can see the usual suspects-(EP)ROM (16 KB), RAM (1 KB), timers, UARTs, and so on. As mentioned earlier, most of this stuff is unchanged from the '5 1. For those of you who aren't familiar with the original, let's take a few moments to review the peripherals and highlight the changes.

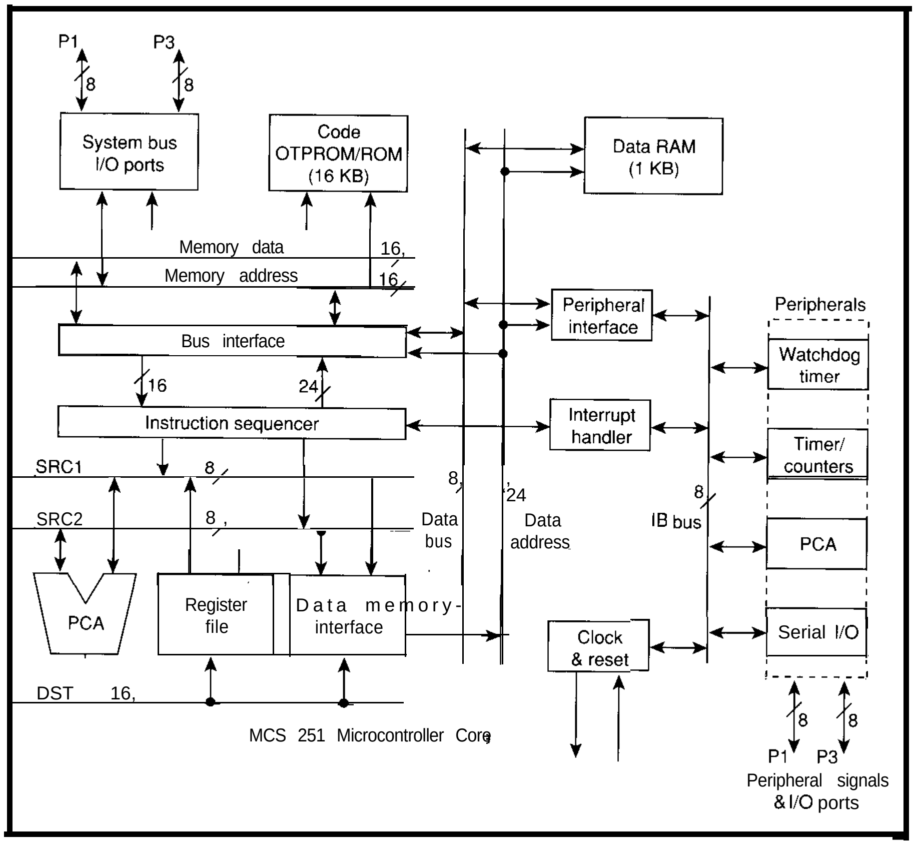


Figure 2—The '251SB combines familiar '51 peripheral functions with extra memory and a brand new CPU core.

There's not a heck of a lot to say about the clock and reset unit, which connects a crystal or oscillator via XTAL1 and XTAL2. Eschewing the "more MHz is better" trend, the maximum clock rate is a leisurely 16 MHz. As you'll see, the '25 1 SB gets its speed by doing things in fewer clocks-not hard since the '5 1 takes a whopping 12 clocks per instruction—rather than simply cranking the clock. Fewer MHz eases interfacing, reduces power consumption, and keeps the FCC at bay.

Note that the '25 1 SB is a fully static design. The clock can run at an arbitrarily slow rate or even be stopped to reduce power from the 60 mA (@ 5 V ±10%) required for full speed (16 MHz) operation. In IDLE mode (12 mA), the clock to the peripherals, but not CPU, is stopped while POWERDOWN mode (20 µA) freezes everything until an interrupt or RESET. A handy feature for the latter case is the addition of a POF (Power Off) flag that detects the difference between a cold and warm RESET.

As for the interrupt logic, the '5 l's original two-level priority scheme is cleverly (and software transparently) extended to four levels by the addition of a second interrupt-priority register (see Figure 3). For compatibility's sake, the '25 1 SB interrupt sources have the same priority and vector addresses as their '51 counterparts. What's new is the addition of a TRAP instruction, which is given highest priority, and the previously mentioned needs-an-extra-pin NMI, which has second highest priority).

The '25 1 SB has plenty of timers, a grand total of nine compared to the two (TO and T1) on the original '5 1. Like many earlier derivatives, the '25 1SB adds a third timer (T2) similar in form and function to the other two. Note the decision to retain the historic time base ($F_{OSC}/12$), thereby favoring compatibility at the expense of resolution. However, when configured as a baud-rate generator, T2 can run at $F_{OSC}/2$ to handle speedy serial links.

Next up is the watchdog timer (WDT), a 14-bit counter, which also increments at $F_{OSC}/12$ and RESETs the CPU if it overflows. Note that the

WDT continues operating in IDLE mode and stops in POWERDOWN mode, either or both of which may be problematic depending on your situation. Keeping the WDT at bay during IDLE can be accomplished by dedicating a timer interrupt to servicing it. If your concern is a POWER-DOWN-induced coma, consider an external power-monitor chip that drives INTO, INT1, or RESET. The WDT powers up disabled, should you prefer not to be bothered. Once you enable it, it remains ever vigilant and can't be shut off.

The last five timers are packed into the PCA (Programmable Counter Array), a unit seen on the earlier 'FX derivative of the '5 1. The PCA, pictured in Figure 4, consists of a counter module that feeds a common clock to five 16-bit register and compare modules, which are in turn connected to the CEXO-4 pins. Each of the five channels can be configured as input capture, output compare, PWM, or software timer. The last channel (module 4) can also be configured as a watchdog timer, which resets the CPU on overflow. But, unlike the fixed 14-bit WDT, it lets you decide the watch period.

The decision to keep the '5 l's leisurely clocking for TO-T2 is made more palatable by the programmability of the PCA clock. Besides the $F_{OSC}/12$ used for the older timers, the PCA also offers $F_{OSC}/4$, an overflow from TO, or an external clock input (ECI) as time-base alternatives.

The UART is little changed from the somewhat homely, but quite serviceable '5 1 unit. One welcome addition is framing-error detection, though the continued lack of parity support seems a little miserly. Also, if T2 is used for the baud-rate generator, the transmitter and receiver can run at different baud rates (on the '5 1, they must be the same). Otherwise, the '25 1 SB UART supports, as in the original, shift register (Mode 0), 8N1 (Mode 1), and the so-called "ninth data bit" modes (Modes 2 and 3).

For those not familiar with the ninth data bit, it's kind of a 9N1 format in which the extra bit can be set or reset by the programmer (for transmission) and mask the normal receive interrupt upon reception. Typically, the ninth bit is used to differentiate between address and data transfers, cutting overhead on a shared bus since nonaddressed micros need not be bothered with data destined for others. Since many other popular micros ('180, 'HC11, etc.) support the ninth data bit, it's possible to lash a surprisingly powerful, yet low cost, network together. One company, Cimetrics Technology, even offers 9-bit-solution software that can network up to 250 micros of half a dozen types with no muss and no fuss.

Having peeled away the layers of normalcy, we finally arrive at the CPU core. It's here that the '251SB earns a place in the X-Files (file under "Processor" right after "Pod People").

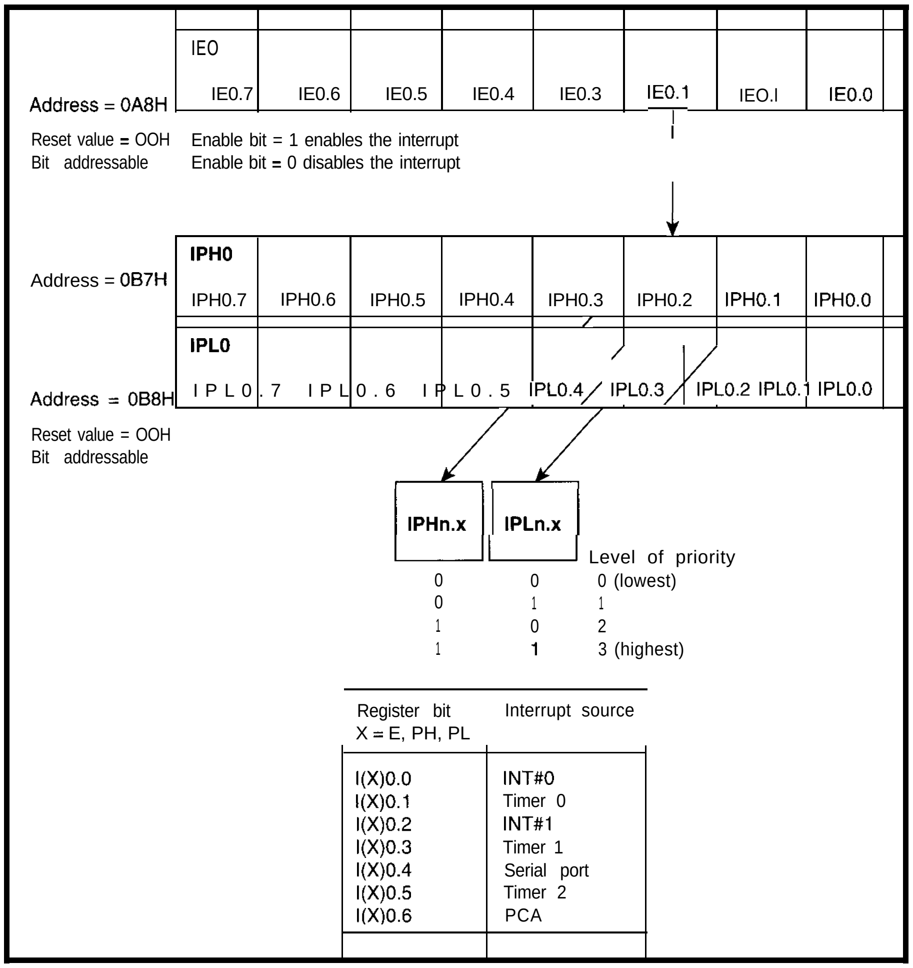


Figure 3—*The '51s two-level, interrupt-priority scheme is extended to four levels in the '251SB. Though the architecture makes provision for NMI, it isn't included for lack of an extra pin.*

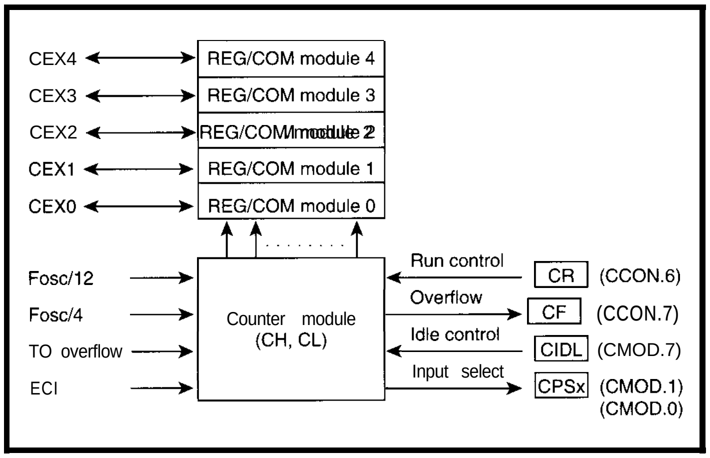


Figure *4-First appearing on the '51FX derivative, the PCA (Programmable Counter Array) features* five capture and compare registers driven by a programmable fimebase

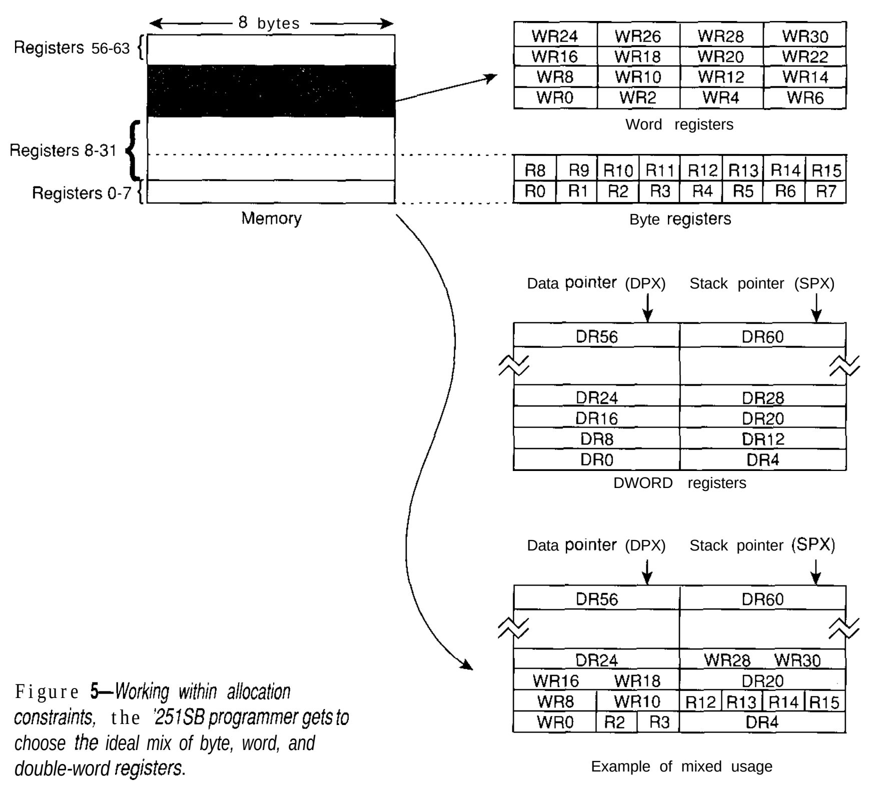


Figure 5—*Working within allocation constraints, the '251SB programmer gets to choose the ideal mix of byte, word, and double-word registers.*

ALIEN OPCODES

It's not hard to jot down a UFO wish list. First, there's the simple question of speed (or lack of in the case of the '5 1}. As mentioned earlier, binary compatibility dictates that old code must be decipherable. At the same time, pressure from competitors, the trend towards C, and a moral obligation to overworked programmers calls for an instruction-set makeover. As for all 8-bit upgrade chips, the 64-KB question, like Bela, lurks waiting to bite.

All in all, it's a tall order, but one I say Intel has carried off with great panache.

The speed issue is addressed by a three-stage pipeline design (fetch and decode, address and read, execute and write) that boosts performance while remaining programmer friendly (i.e., no visible pipe hazards).

Though the ALU and external bus remain 8-bit, the '25 1 SB keeps the pipeline fed by fetching instructions, 16 bits at a time, from on-chip EPROM.

While the '5 1 executes the bulk of its instructions at the same rate (12 clocks), '25 1 SB instructions are of variable duration. After all, it doesn't make sense that an on-chip register op should have to take as long as a memory move. Overall, the '25 1 SB probably executes existing '5 1 instructions on the order of three to five times faster than the original.

While supporting '5 1 instructions, the '25 1 SB surrounds them with a bunch of new opcodes, addressing modes, and registers that together attack the program friendliness and 64-KB concerns.

As expected, the first task is to eliminate the '51's infamous accumulator bottleneck by adopting a general-purpose register architecture. Besides 32 bytes replicating the '5 1's registers, an additional 32 8-bit registers are available for use as bytes, words (16 bits), or even double words (32 bits]. The latter, when used as pointers, are the magic bullet for the 64-KB problem.

The allocation and naming rules may seem a little odd at first, but do offer the flexibility to divvy things up just the way you want (see Figure 5). Registers O-15 can be addressed as either byte, word, or double word; registers 16-3 1 as either word or

CIRCUIT CELLAR INK®

ECHNICAL
OSTCARD
DECK

SPRING '95
EADLINE
DATES

SPACE: 4/7
ATERIAL: 4/13
MAIL: 5/8

GREAT RATES
RESPONSIVE READERS
FREE READER SERVICE

CONTACT
SUE HODGE
(203) 875-2199
X (203) 872-2204
YOUR AD REP
TODAY.

Config0 (Address 80H)

Symbol	—	—	WSa	XALE	Rdl	Rd0	PSwap	Src

Config1 (Address 81 H)

Symbol	—	—		Intr	WSb	—	—	EMap
Bit	7	6	5	4	3	2	1	0

Symbol	Function
	Reserved
Src	Source/binary mode configuration
	Device is only source-code compatible
0	Device is also binary-code compatible
PSwap	Swaps the Data bus from Port0 to Port2 and implements a page-mode functionality
	The Data bus is set to Port0
0	The Data bus is swapped from Port0 to Port2 and implements the page-mode functionality
Rd0-1	Configures Rd# and PSEN# to alternate functions
	Rd# function
1 1	Rd# strobes for address <80:0000H
1 0	P3.7 only
0 1	Rd# is Al 6
0 0	Reserved
XALE	ALE extension configuration
	Does not extend ALE
0	Extends the ALE pulse from Telel to 3 Telels, which adds 1 wait state
WSa	Main external wait state configuration
	Does not generate any extra wait states
0	Generates 1 extra wait state for all pages except the 01 : page
EMap	EPROM mapping configuration
	Does not map the external code memory
0	Maps the upper 8 KB of the 16 KB internal code memory into addresses 00:E000H–00:FFFFH.
WSb	Page 01 : external wait state configuration
	Does not generate any extra wait state
0	Generates 1 extra wait state for the 01 : page
Intr	MCS251/MCS51 interrupt mode configuration
	Interrupts push 4 bytes on the stach (3 bytes of PC & PSW1)
0	Interrupts push 2 bytes of the stack ala the MCS51 (64 KB execution restriction)

Figure 6—*The '251SB is a switch hitter when it comes to binary versus source compatibility with the choice made by the Src and Intr configuration bits.*

double word; and registers 56-63 are double word only.

Thus, the architecture supports 16 possible byte registers (RO-R15), 16 possible word registers (WR0–WR30), and 10 possible double-word registers (DRO-DR28, DR56–60). Note that the numbering scheme points a register at the first byte (i.e., R4, WR4, and DR4 all start at the fourth byte). The flip side of this is that sequential register addresses increment by one for bytes (i.e., RO, R1, R2), two for words (i.e., WRO, WR2, WR4), and four for doubles (i.e., DR4, DR8, DR16).

Note how DR56 and DR60 map to extended versions of the '5 l's DPTR and SP. Yes, the '5 l's stack, whose tininess and inaccessibility are legend, is finally replaced with something a C compiler can learn to love.

Without going into all the detail, suffice to say that the new instructions, addressing modes, registers, and so on make up a relatively clean architecture that can deal with l-, 8-, 16-, and 32-bit work. There are some restrictions about mixing old and new (e.g., old instruction with new addressing mode and vice versa), but they seem neither illogical or especially painful.

The '25 1 SB cleverly deals with the tradeoff between code compatibility and optimizing the new instruction set. How? By letting you deal with it.

The original '5 1 designers were kind enough to leave one spare opcode (A5h). The only way to keep binary compatibility is to map the new instructions using the spare opcode as an escape prefix. The problem, of course, is that your shiny new instruction set carries extra baggage on every I-fetch, so code density suffers.

For those who are writing new programs (or at least can recompile their old ones), it makes more sense to remap the opcodes, giving priority to the new and improved instructions, while hanging the prefix on some of the older, less useful ones.

Which way is best? The answer-depending on the circumstances, both. Figure 6 shows the key configuration info that determines the '25 1SB's personality. The Src bit chooses between the binary and source-compatible modes. Note that these bits are programmed into the '25 1 SB EPROM for automatic loading at RESET.

Similarly, the Intr bit decides whether the CPU acts like a 64-KB or 16-MB address-space machine. In 64-KB mode, it mimics a '51 by stacking only 16 bits of the PC during an interrupt. In 16-MB mode, 24 bits of PC and the PSW are stacked. Note that the CPU maintains two PSWs: one that's exactly the same as the '5 1 and one supporting the new instruction set.

As you can see in Figure 7, the other configuration bits are mainly responsible for tuning the external bus interface. Note the minimum cycle time differs for read and write (four and six clocks, respectively). If you prefer, you can set it up with wait states to act just like a '5 1 and take advantage of other neat options.

One option is to swap the multiplexed data from the low to high address byte, which simultaneously implements a page-mode function. As

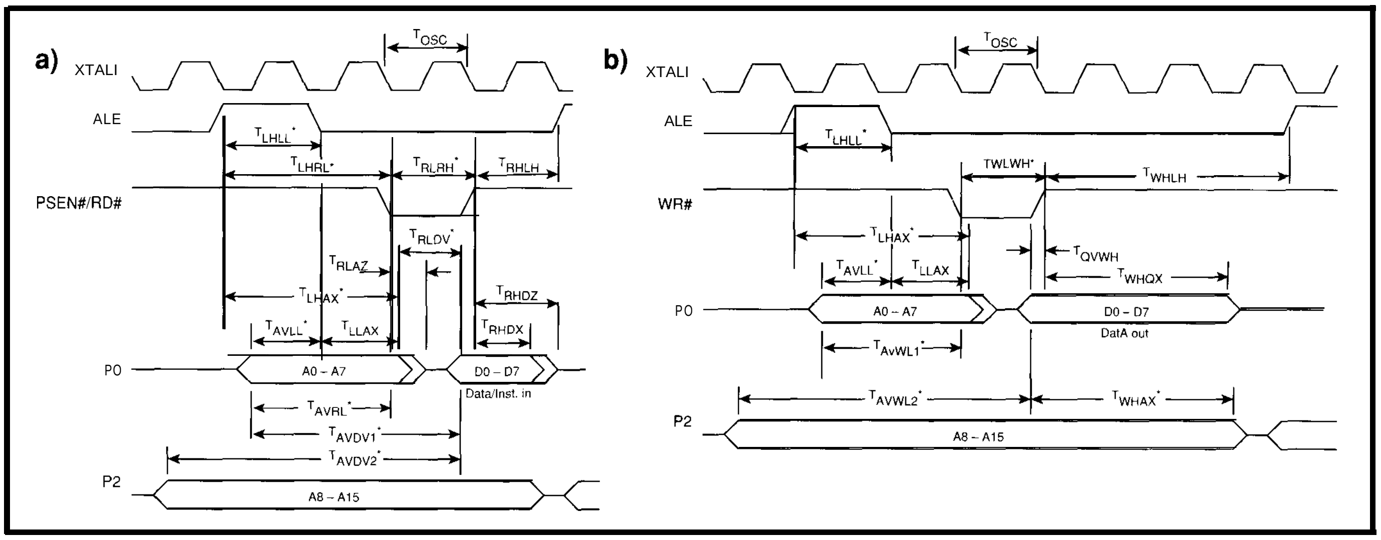


Figure 7-Here are the nonpage modes for the data/instruction read cycle (a) and the write cycle (b). Tne '251SB bus cycle can be extended by up to four clocks, two during ALE and two during the read or write strobe. When the page-mode option is chosen, ALE cycles are only issued when the high address changes.

you might guess, this scheme offers speedy (no-ALE) accesses within a page (until the high address changes).

Being able to assign the function of the RD * and PSEN * (Program Space Enable) pins is a nice touch. Naturally, you can make them work the same as a '51 (i.e., RD' is for data and PSEN* for instructions). Another choice duplicates the common practice of overlapping them (using PSEN* only), which frees a pin (RD*) for use as either a port bit or an address line.

LOW BUDGET THRILLER

Admittedly, the '251SB can't offer the IC equivalent of the big names or grandiose sets of Hollywood's latest epics. Just remember, most of the rough edges (no NMI pin, limited memory space, etc.) are an unavoidable byproduct of the reasonable decision to deliver binary and plug compatibility.

No doubt bigger-budget sequels with more pins and transistors are already in the works. Unlike Plan *10 From Outer Space,* I am certainly looking forward to Plan 252. In the meantime, with the '25 1 SB price projected at $6 (ROM) in high volume, you'll have plenty of change for popcorn. ❑

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

CONTACT

Intel Corporation
5000 West Chandler Blvd.
Chandler, AZ 85226
(916) 356-3551

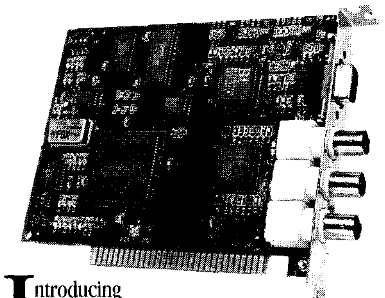
Cimetrics Technology
55 Temple Place
Boston, MA 0211 l-1300
(617) 350-7550

I R S

422 Very Useful
423 Moderately Useful
424 Not Useful

PRECISION FRAME GRABBER

FOR ONLY $495*



Introducing the CX100 precision video frame grabber for OEM, industrial and scientific applications. With sampling jitter of only ±3 nS and video noise less than one LSB, ImageNation breaks new ground in imaging price/performance. The CX100 is a rugged, low power, ISA board featuring rock solid, crystal controlled timing and all digital video synchronization.

Software developers will appreciate the simple software interface, extensive C library and clear documentation. The CX100 is a software compatible, drop-in replacement for our very popular Cortex I frame grabber. Call today for complete specifications and volume pricing.

— CXIOO FEATURES —

- Crystal Controlled Image Accuracy
- Memory Mapped, Dual-Ported Video RAM
- Programmable Offset and Gain
- Input, Output and Overlay LUTs
- Resolution of 5 12x486 or Four Images of 256x243 (CCIR 512x512 & 256x256)
- Monochrome, 8 Bit, Real Time Frame Grabs
- Graphics Overlay on Live or Still Images**
- External Trigger Input
- RGB or B&W, 30 Hz Interlaced Display
- NTSC/PAL Auto Detect, Auto Switch
- VCR and Resettable Camera Compatible
- Power Down Capability
- BNC or RCA Connectors
- Built-In Software Protection**
- 63 Function C Library with Source Code
- Text &Graphic Library with Source Code
- Windows DLL, Examples and Utilities
- Software also available free on our BBS
- Image File Formats: GIF, TIFF, BMI? PIC, PCX,TGA and WPG

ImageNation Corporation
Vision Requires Imagination
800-366-9131

** THESE OPTIONS AVAILABLE AT EXTRA COST
* $495 IS DOMESTIC, OEM SINGLE UNIT PRICE.

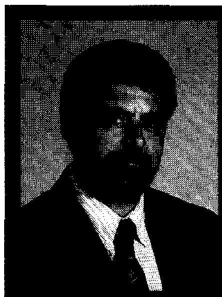
P.O. BOX 276 BEAVERTON, OR 97075 USA PHONE (503) 641-7408 FAX (503) 643-2458 BBS (503) 626-7763

#133

EMBEDDED TECHNIQUES

John Dybowski

Embedded Development



Program development for small embedded processors can be problemmatic without the proper tools. John's answer—he emulates the AT89C2051 with the 8051-compatible AT89C51, saving lots of time and money.

embedded systems vary tremendously in function, capability, and complexity. In fact, they have no single discernible characteristic. It might be simpler to explain these systems by what they don't possess and can't do.

In many cases, these systems are so specialized and of such dedicated purpose that it may be impossible to pick them out of the background. Sometimes, they have no user interface, produce no perceptible output, and have no power switch. As a result, it's not surprising that at times it's impossible to tell if they're running.

These attributes (or more accurately, lack of attributes) make designing and debugging such equipment extremely difficult. Obviously, developers of such systems need some outside help since the amount of information they can glean from an unaided examination of this type of equipment is sparse indeed.

Engineers use a variety of tools when developing microprocessor- and microcontroller-based systems. Not unexpectedly, the cost of these tools increases as more features and more capabilities are added. What comes as a surprise is how much the cost of a truly deluxe development system exceeds that of a merely adequate one.

This disproportional phenomenon predominates in all facets of the engineering discipline: 10% of the code runs 90% of the time, 30% of the program delivers 95 % of the functionality, and the last 2% of the project takes considerably longer than 2% of the overall time. The numbers vary, of course, but it always boils down to the same imbalance.

A lot of different tools can be applied to microprocessor-based development. These run the gamut from EPROM emulators to full-blown in-circuit emulators. The right choice depends on what you're trying to do, how much time you have, and perhaps most importantly your budget.

The fact is, even with a full-featured development system, you'll probably use only a fraction of its capability most of the time. It's only those (hopefully rare) times when you have a really tricky problem that you tap the full potential of the system. But at these times, you don't mind its price tag either.

A special set of problems arises when working with very small processors. The situation is exacerbated when the degree of integration reaches that of a system on chip (i.e., a microcontroller). Needless to say, it creates rather awkward debugging scenarios.

In most cases, you debug out of the microcontroller's EPROM.

Mode (1)	RST	P3.2	P3.3	P3.4	P3.5	P3.7	P1.0–7 I/O
Write code data	12 V	0.1 ms	L	H	H	H	DI
Read code data	H	H	L	L	H	H	DO
Write lock bit-1	12 v	0.1 ms	H	H	H	H	FF (Hex)
bit-2	12 V	0.1 ms	H	H	L	L	FF (Hex)
Chip erase	12 V	10 ms	H	L	L	L	x
Read Atmel code	H	H	L	L	L	L	DO
Read device code	H	H	L	L	L	L	DO

Table I-Various *f/ash and configuration functions can be invoked by manipulating fhe I/O pins on the A T89C2051*

You change the source file, compile and link, burn the EPROM, plug into the target system, apply power, and-nothing!

Obviously, you don't want to repeat such unproductive steps too many times.

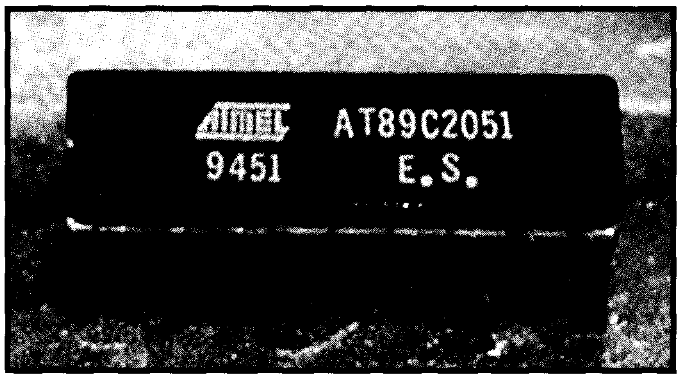


Photo 1—*The AT89C2051 is just like an 8051, but smaller.*

SMALL PROCESSORS

In addition to these difficulties, small processors are often cursed with "weird architecture." Mind you, what we consider weird today may be perceived entirely differently with time.

Since I delight in the AT89C-205 l's rigorous adherence to the 805 1 standard, the story comes full circle. Surely no one would dispute that the 805 1 is mighty weird in its own right. However, engineers have been working with this processor so long they no longer notice!

In any event, working with certain small processors presents unsavory problems. A proprietary architecture could leave you essentially debugging out of EPROM for lack of any affordable support equipment. Worse, you might find yourself hostage to the chip manufacturer's idea of a tool set.

Luckily, the situation changes significantly when considering a "standard" architecture. Things look even better with "subset" processors provided they retain and give up the right features of their predecessors.

As a full 8051 implementation, the Atmel AT89C205 1 (see Photo 1} gives up some I/O and comes with 2 KB of flash program memory instead of the usual 4-KB EPROM. Despite its 20-pin package, the 89C-205 1 retains the major 8051 SFRs and 128 bytes of internal RAM. The importance of the full 128 bytes of RAM will be immediately obvious to anyone who's used other small 805 1 spinoffs with only 64 bytes. This extra memory spells the difference between using a language compiler or working entirely in assembler.

Since the AT89C205 1 uses flash technology for its program storage, the resulting flexibility can be put to good use during program development. For simpler applications, this chip with a programmer may be all that's needed.

However, there are mitigating circumstances that favor a more flexible design environment regardless of project size. Despite what might be considered "correct" design procedures, developing embedded applications is an iterative process. The method of implementing features, eradicating bugs, and enhancing performance is interactive by nature.

This is especially true in the realm of small system development. Here, the existence of the many interdependencies and entanglements that exist in a typical application program inevitably leads the programmer to one of the fundamental tenets of the small-system codesmith: "Change only one thing at a time. Regardless of what they tell you." Experience shows that this is the prudent course of action.

Certainly, modern computers make the edit, compile, and link iteration fast and painless. Unfortunately, the problem with developing out of an EPROM-based system is the amount of effort required to put the program into the chip. This development method is therefore usually prohibitive unless the application is very simple or you've got a lot of time.

A REASONABLE APPROACH

As I've shown with some of the other systems, combining a PC-hosted debugger with a resident kernel running on a small RAM-based computer can result in a potent, low-cost development environment. As luck would have it, the architectural compatibility between the small Atmel processor and its larger predecessors makes this approach feasible.

It's relatively straightforward to use an 805 1 with external program RAM to emulate the smaller derivative. How much you monkey with your code for the final transition to the ultimate target processor depends a lot on how much license the chip designers took with the derivative's resources.

These resources are in effect what the processor is made of-the timers, interrupts, SFRs, program and data memory, and the external I/O ports. In all these respects, the 89C-2051 is "really close" to the original 805 1.

But, it's more than just a hardware compatibility issue. As is so often the case, the software is the system. Since this is essentially an embedded project, it boils down to a matter of software and firmware.

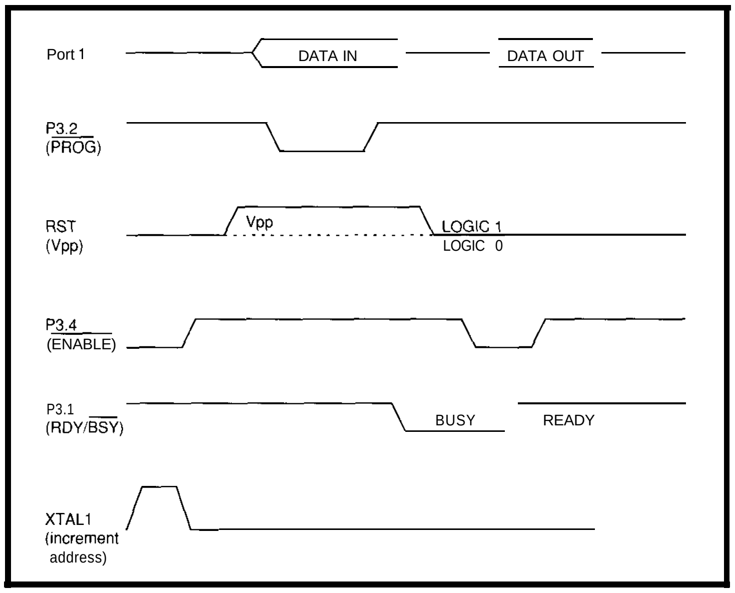


Figure l--The *AT89C2051's flash* memory *programming cycle is self-timed and is controlled using several port 3 pins.*

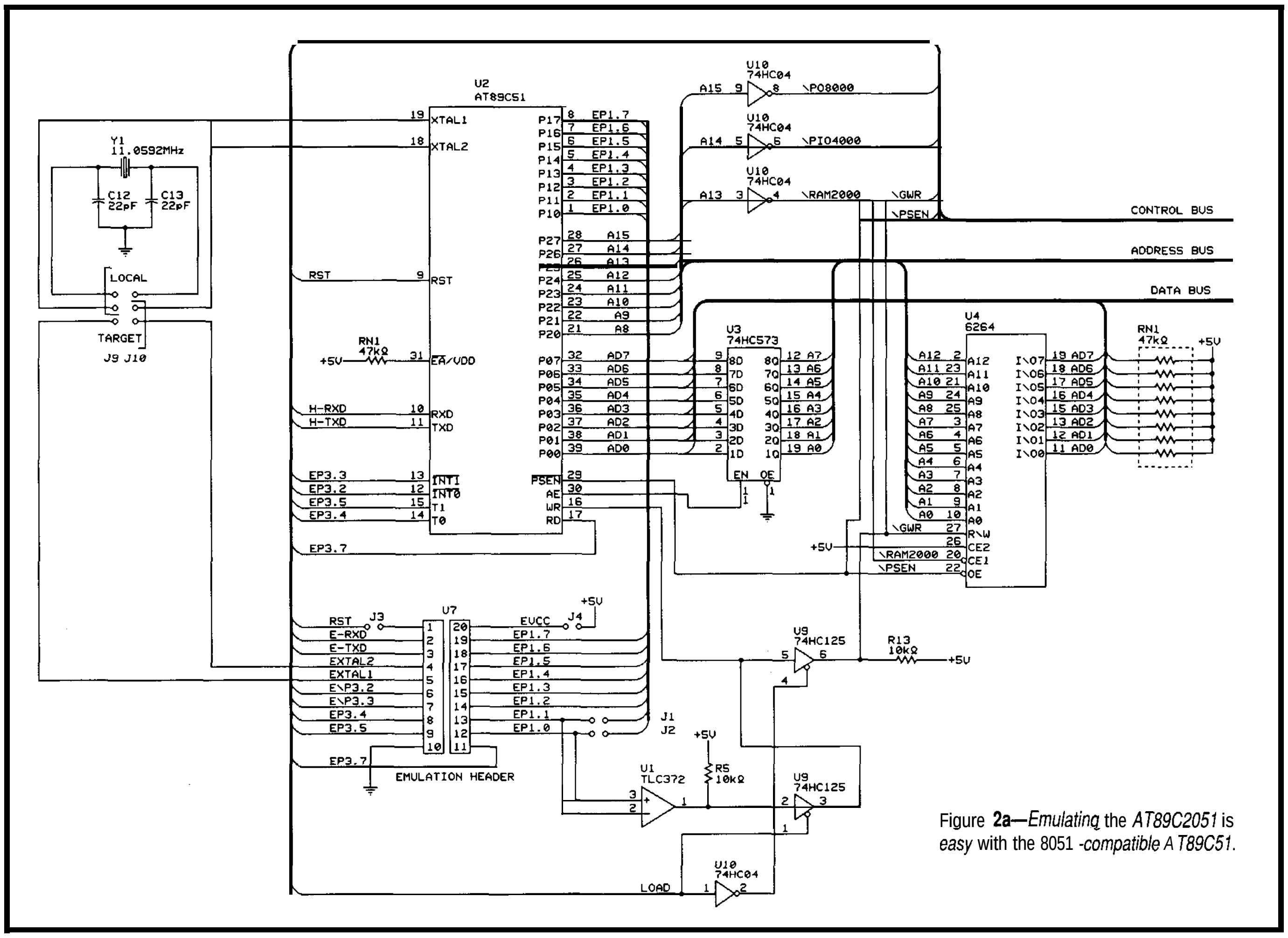


Figure 2a—*Emulating the AT89C2051 is easy* with the 8051 *-compatible A T89C51.*

First off, to support a flexible debugger, the executable program loads and executes from RAM. This permits quickly downloading programs, setting breakpoints, single stepping, and manually modifying program memory. Additional debugger functions provide a gateway to internal workings of the processor. It should be easy to display and manipulate all SFRs, internal memory, and I/O ports.

Of course, a good debugger should be as unobtrusive as possible. To achieve this goal, my development system runs under the Dunfield debugger system with various Mid-Tech extensions. The tiny (less than 2 KB) firmware debug kernel off-loads the actual data processing tasks to the PC-resident debug monitor. It simply performs the lowest-level tasks as directed by the PC control program.

This distributed processing is absolutely necessary since the goal is to stay out of the way of the system under development. Putting the real processing on the PC where the power is results in a friendly windowed environment from which you can wage your debug session with minimal target impact.

Of course, there's more to being unobtrusive than just keeping the kernel small. Consider this: the Dunfield kernel uses zero internal memory, zero stack, and has no effect on any internal processor registers or SFRs. How's that for unobtrusive?

The PC-hosted monitor provides the interface into the target processor and therefore the system under development. This interface is identical to that of the compatible, stand-alone AT89C2051 simulator. Together, these tools let you bring your application up gradually.

The simulator lets you dry-run your algorithms using your PC. If you desire, you can run your code simulation on the PC while using your target system's real I/O via the development system interface. Eventually, you can move to full target emulation. The final step is to burn the program into flash and run entirely from the 89C-2051.

These steps let you start development in the safety of your PC and move your application gradually to the Spartan realm of the single-chip controller. This greatly increases the chances of a functional 89C205 1 when you get around to programming it.

PROGRAM MEMORY IN A FLASH

The AT89C205 1 contains 2 KB of flash memory. Atmel calls this memory array *PER OM* (Programmable Erasable Read Only Memory).

The flash-memory array is sectored into one monolithic 2-KB block. The memory is programmed byte-by-byte by raising the RST/VPP pin to 12 V and presenting parallel data on the 8 bits of P1. P3.2 functions as the programming strobe by pulsing low, and data can be read for verification purposes by pulling P3.4 low as a data-enable signal. The progress of the self-timed programming cycle can be interrogated using data polling, or alternatively P3.1 can be used as an active-low, program-busy indicator.

In combination with these control lines, pins P3.3, P3.5, and P3.7 set the various modes of operation. These modes include programming the two internal lock bits, reading the device code ID, and programming, reading, and erasing the memory.

We've washed up most of the I/O pins and haven't even considered addressing the memory array yet. Evidently, there must be a way to efficiently handle address generation. The 89C205 1 contains a built-in address generator that sequences the internal address using an external clock input.

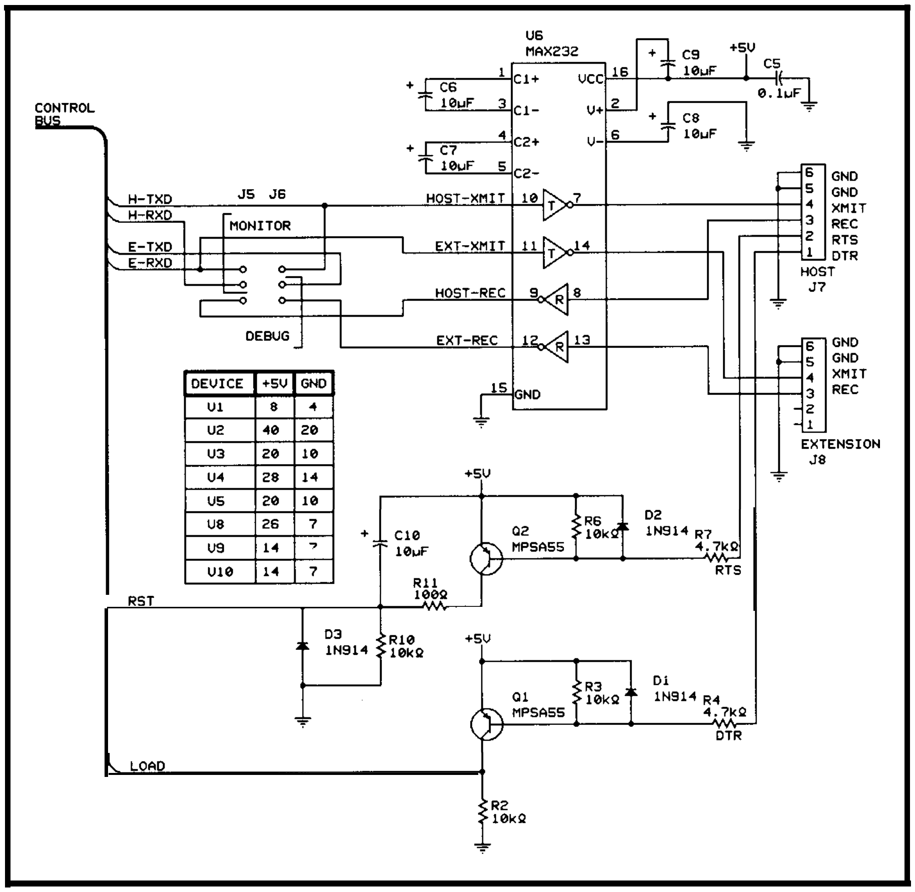
Here's how it works. When RST/VPP is initially held at logic 0, the internal address generator is reset to 000. The address is advanced by applying a positive pulse on the XTAL1 pin. This address is now stable and the specified location can be accessed for programming and verification.

The byte programming cycle is self-timed. There are two methods that can be used to determine when the cycle completes. While a program cycle is executing, an attempt to read the last byte programmed results in the complement of the programmed data on P1.7.

Once the program cycle has completed, the actual data is available on all outputs. The program cycle progress can also be monitored directly using P3.1. This pin indicates a busy condition while it is low. It goes high when the programming cycle is complete and the next location can be programmed.

These steps are fairly straightforward, but prior to programming any nonblank memory bytes, you need to perform a bulk-erase operation. That is, the entire memory array must be set to FFs before any individual bits are programmed to 0.

This erasure is performed electrically by setting the proper combination of control signals and pulsing P3.2 (the programming strobe) while RST/VPP is held at 12 V. The bulk-erase operation is self-timed and takes about



DEVICE	+5V	GND
U1	8	4
U2	40	20
U3	20	10
U4	28	14
U5	20	10
U8	26	7
U9	14	7
U10	14	7

Figure 2b—*The* PC interface provides *full* control of the *development system.*

10 ms. Table 1 depicts the way the control lines are manipulated to perform the erase, program, verification, and configuration functions. Figure 1 shows the waveforms for a typical program and verify cycle.

A REAL DEVELOPMENT SYSTEM

The AT89C205 1 development system consists of several hardware, firmware, and software components. These include a development card with a RAM-based 89C205 1 emulator and flash programmer, the firmware-resident debug kernel and flash utilities, and a PC-hosted simulator, debugger and assembler.

Although the development card could be powered by a standard 805 1 (or 803 1 with EPROM), I decided to use the Atmel AT89C51. This is essentially a CMOS flash-based 8051, which maintains strict compatibility with generic 805 1 architecture.

To those experienced in the development of very small embedded systems, the problems associated with such an undertaking are well understood. It makes little sense making things more difficult than necessary. Building on the foundation you have can cut time out of your design phase.

The close architectural association of the 89C2051 and the standard 8051 makes migrating 805 1 applications to the new small processor fairly easy. Since a microprocessor development system is nothing more than a specialized embedded system, it follows that similar benefits can be realized by porting an existing tool set to support a new processor.

The system components include the main development card with flash programmer and a small, general-purpose target card that can be used with the development system or with a real 89C205 1 processor. The main development card is depicted schematically in Figure 2.

As you can see, this system is built around Atmel's flash-based 89C51 and follows the lines of the more conventional 805 1 design with external RAM. The close and nonconflicting correspondence between the

89C205 1 and 805 1 results in minimal extraneous hardware. All of the P1 and most of the P3 lines directly carry through to the emulation header.

The analog comparator contained within the 89C205 1 is simulated using the externally located TLC371. On the 89C205 1, the output of the built-in analog comparator internally ties to P3.6. Neither the comparator's output nor P3.6 is externally accessible. On the 89C51, P3.6 is a general-purpose port pin and also functions as the external RAM \WR strobe. Using discrete logic, the comparator is degated from P3.6 when the host downloads or in any way manipulates the external RAM. During these operations, the 89C51 uses P3.6 as a write strobe.

The host PC signals the development card that it is seizing system memory by asserting its DTR. This action illustrates the close coupling between the host PC and the development card. But, make no mistake about who's in control. Using RTS, the PC can exercise ultimate control and yank the development system (and therefore, the target) back to square one really fast-RTS directly controls the master reset to the system.

The development card lets you use P1 .O and P 1.1 as general-purpose I/O bits or as high-impedance inputs to the analog comparator. To preserve the high-impedance characteristic of the analog inputs, two shorting jumpers disconnect the P1 .O and P1.1 header signals from the 89C51's respective I/O lines. Rather than emulating the 89C205 l's electrical I/O characteristics completely, the jumpers let you simulate the functionality of the 89C205 1 without complicating the hardware design.

No attempt is made to directly simulate the high-current-carrying capability of the 89C205 1 general-purpose port pins. Many applications do not use this special capability. If it's desirable or necessary to provide heavier current sinking directly under emulation, it can be handled on a need-to-do basis. Using external circuitry, any line requiring extra sink capability can easily be equipped with an outboard driver.

Although the flash-programming subsection could have been designed using the same 89C5 1 port pins that emulate the target 89C2051, a dedicated set of memory-mapped I/O ports is used for this function. This gives greater control and imposes fewer constraints than is feasible if the 89C5 l's I/O ports are shared between the programming fixture and the target system.

The main I/O signals are derived from an 82C55 programmable peripheral interface. Here port A is used as an 8-bit parallel-input port for reading the 89C2051. Port B drives the logic-level control signals to the programming site. Finally, port C handles the main system controls, including enabling VCC to the chip, driving 0, 5, or 12 V into the RST/VPP pin, and enabling the 74HC374 octal flip-flop. The flip-flop serves as the memory-mapped, tristate, parallel-data output port.

When the 82C55 emerges out of reset, all ports default to inputs. Once the chip is programmed to its desired operating mode, any ports selected for output immediately begin emitting zeros. To avoid glitching any outputs, the control logic driven by these outputs is set up to live with the chip's idiosyncracies (i.e., the idle state of all important outputs is defined as a logic low).

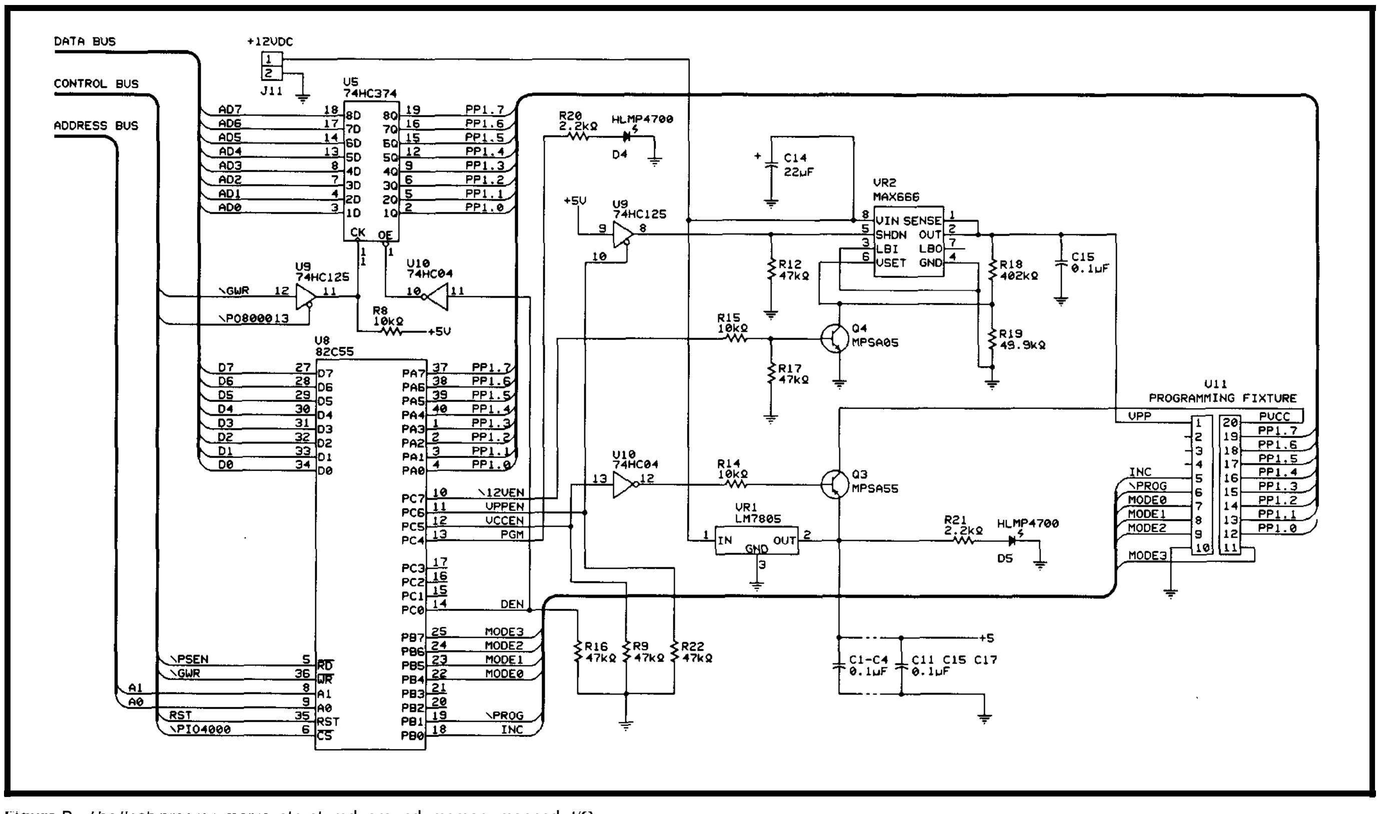


Figure Pc-*The flash programmer is* structured around memory-mapped *I/O*

Pull-down resistors on these outputs ensure that these pins are held low from reset through the subsequent chip initialization steps. Because of the way the 82C55 resets its outputs each time it is reprogrammed, I elected to add an external output port rather than "turning around" port A for both input and output. There's no sense getting overly clever and closing the door on flexibility.

With the development system in program mode, all lines to the ZIF flash fixture are placed into a quiescent state. The interface is primarily composed of direct connections between the 89C2051, 82C55, and 74HC374, which constitutes a logic-level interface. Additionally, a processor-switched VCC connection is provided along with the ground return.

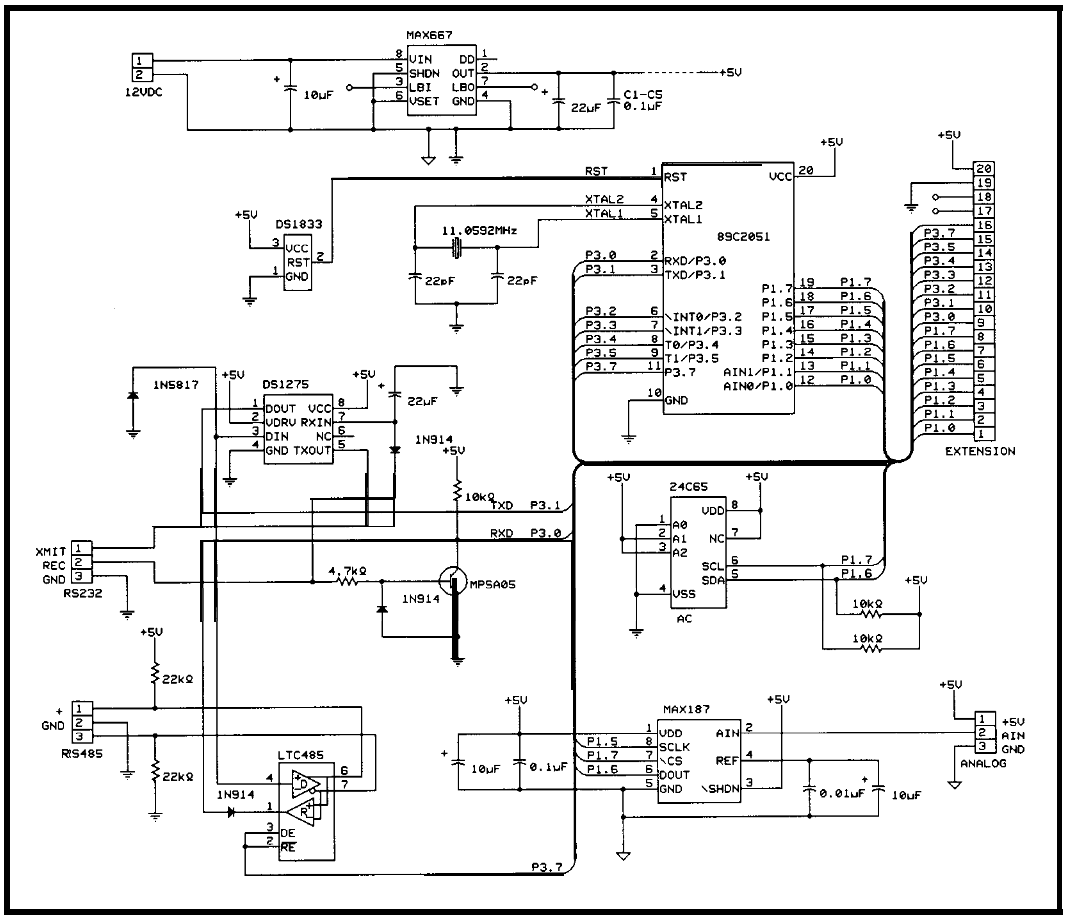


Figure 3—The AT89C2051 single-board computer includes a power supply, line-powered RS-232 interface, 8 K by 8 EPROM, and a 12-bit ADC

The only "special" voltage on the programming fixture is brought out to the RST/VPP pin. The trilevel signal on this pin can be pulled to 0 V during chip initialization, set to 5 V for read and verify operations, or driven to **12** V for byte programming and bulk-erasure modes.

To accomplish this, a MAX666 micropower pass regulator is configured to deliver either 0, 5, or 12 V. Pulling the SET pin to ground programs the regulator for a fixed 5-V output. Releasing the ground connection effectively enables the resistor divider that selects a 12-V output level. The regulator can be disabled by driving SHDN high. In this case, the output is resistively pulled to ground to establish the logic 0 level.

The MAX666, although an older design, serves this application well since a minimum number of external components and control lines establish the required voltage levels. Being a low-dropout regulator, the MAX666 is capable of maintaining proper regulation with only a 200-mV input/output differential. And, unlike some low-dropout designs, only a very small output capacitor is required. This speeds up switching speed considerably.

REAL EMBEDDABLE

Figure 3 shows the AT89C2051 single-board computer. This standalone system includes a power supply, line-powered RS-232 interface, 8 K by 8 EEPROM, and a 12-bit ADC. Expansion is available in the way of the I^2C peripheral set I presented last month. A subsystem card attaches directly to the SBC, which includes an RTC, RAM, battery backup, EEPROM, and digital and analog I/O. An outboard LCD and keypad module provides an avenue for a user interface panel.

Obviously, the attraction of such a system is not so much what it is or even what it can do as much as where you can put it. For this reason, there's not much I can say about the system itself until I present a real embedded application for the system. And, at 1" by 3", it can get real embedded.

John Dybowski is an engineer involved in the design and manufacture of embedded controllers and communications equipment with a special focus on portable and battery-operated instruments. He is also owner of Mid-Tech Computing Devices. John may be reached at (203) 684-2442 or at john.dybowski@circellar.com.

SOURCE

For elements of this project, contact:

Mid-Tech Computing Devices
P.O. Box 218
Stafford Springs, CT 06075-0218
(203) 684-2442

IRS

425 Very Useful
426 Moderately Useful
427 Not Useful

CONNECTIME

conducted by Ken Davidson

The Circuit Cellar BBS
300/1200/2400/9600/14.4k bps
24 hours/7 days a week
(203) **871-1988—Four** incoming lines
Internet E-mail: **sysop@circellar.com**

I'm short on space this month, so I'll limit things to one thread and one followup message.

In the message thread, we fake a look at NiCd batteries and some myths and truths surrounding their use and care. I really wish someone would come up with a low-maintenance, long-lasting, inexpensive rechargeable battery.

In the followup message, a "ConnecTime" reader responds to last month's thread on three-phase motors with his own experiences and suggestions.

Cellular NiCds

Msg#:18940
From: GEORGE COHN To: ALL USERS

I'm sure this subject has been beaten to death, but I'm curious about so-called memory in NiCd batteries. I've noticed that the NiCd battery on my cellular phone only lasts about 6 hours now in standby mode. When it was new (about 16 months ago) it lasted 14 to 18 hours.

What prompts the question is Hello Direct has a cellular phone battery recharger/conditioner that supposedly strips away the bubbles that form on the plates using a negative pulse. They claim that this will bring the old battery back to its original capacity. The cost of this gadget is $99.95. A replacement battery for my phone is $59.95. Obviously it would be advantageous to make the old batteries last longer for both cost and ecological reasons. Has anyone had experience with this type of charger?

Msg#: 19336
From: RANDY **RIDLEY** To: GEORGE COHN

You're right. It has been beat to death! <g> But It has been several months since I've seen a posting on it so I'll give you a quick explanation of how a NiCd works.

First, the "memory effect" is a misnomer created by GE in the early years of NiCd technology. It was a political move to stall the competition's edge while they caught up. Memory effect is only reproducible under strict lab conditions and the average NiCd user will never see it.

What you do see is an effect called "voltage depression." The NiCd's electrolyte is a random fill of packed powder. It must stay in a random orientation to produce the electrical potential. If a NiCd battery is left charging continuously, two things can happen. First, the electrolyte can begin to crystallize and form conductive "chunks" in the battery. Since the total energy stored by the battery is dependent partly upon the separation between the electrodes, the conductive chunk of electrolyte effectively shortens the life of a battery. This is usually a recoverable condition as I will explain later. Second, conductive whiskers tend to grow from the electrodes through the electrolyte. We call these "tin whiskers" but they are generally made up of impurities in the electrolyte or from the electrodes themselves. As the battery is used, the electrodes become pitted with material leaching into the electrolyte. This material can reform in a conductive whisker similar to the old "slat crystal" that you probably made in high school chemistry.

The "voltage depression" effect is caused by leaving a battery in one state for too long of a period. This state can be completely discharged, completely charged, or anywhere in between. It is a result of too little use of the battery. Studies by Motorola Energy Products have shown that a battery (even in a severe state of voltage depression) can be revived by about three complete discharge/charge cycles. By leaving your phone plugged into a charger all of the time, you usually end up suffering from this condition. (Personally, I would rate this as the number one cause of discarded NiCds, but I don't have figures to back that up).

You have seen the industry attempt to combat this with better chargers on the market in the last two years. "Smart" chargers will completely discharge the battery for you and then charge it back up. This type of charger is probably good for the man who has everything or the company that has too many batteries to deal with any other way, but is really unnecessary for the average user. An effective treatment is to use your phone as is comfortable to you. When you see the battery is beginning to deteriorate in performance, charge it up and then leave the phone on until it completely discharges. Perform that two or three times and you will probably see your battery recover just fine. The batteries will never recover to full "new" potential, but they will get close.

If the battery does not respond to that treatment, it most likely has severely pitted electrodes and is toward the end of its useful life. A quality battery under optimum conditions can see 1000 to 5000 complete discharge cycles.

Cheaper batteries will not last as long. Obviously, you can see that a battery charger that discharges the battery every time you use it can really shorten the useful life of your battery. If your battery is in this condition, it may have the "tin whiskers" forming on the inside and you may be able to recover some useful life by shocking the battery.

The idea is to send a high-current pulse through the battery to treat the whiskers as a fuse. You have to have a pulse high enough to burn the whiskers, but low enough so the battery can absorb it without exploding. It's sort of a by-guess-and-by-golly process. If you attempt this, make sure that if the battery explodes, you are protected by some sort of physical shield. Also, never shock batteries in series. You must disassemble the individual cells and treat each one independently. This treatment will only give you a slightly longer life and is very dangerous. It may not be worth it.

Msg#:20257
From: LEE STOLLER To: GEORGE COHN

There's something else that happens, and I think it's more common than the other things you hear. People tend to use their phones for a while, then put them in the charger until the next day. Once the battery charges up, the current going through it in the charger produces heat, which tends to dry out the cells. You can observe this for yourself by taking the phone out of the charger after an hour or so, then feeling the battery with your hand. Then put the phone back, and do the same thing after 12 or 16 hours. You'll feel the heat.

The thing to do is check the phone periodically when it's on charge, and when the LCD display indicates "full," take it out of the charger and leave it out until you've used the phone and are ready for recharge. Don't let the new battery that you buy get hot and it will last much longer.

3-phase motor followup

Msg#:31099
From: WALDO BOYD To: PETE CHOMAK

Your message 36086 mentioned in ***Circuit Cellar INK*** issue 55, plus discussions, was of much nostalgic interest to me. Some years ago I was confronted with the same problem and made many inquiries. Here are three sources for data on the subject of single-to-three-phase conversion sans rotary converters:

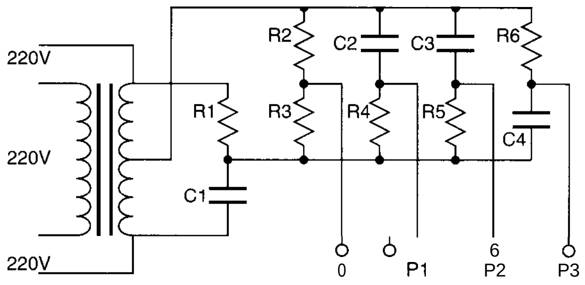
U.S. Dept. Agriculture, Farmers Bulletin No 2252, March 1972, via Supt of Documents, Washington D.C., entitled "Phase Converters for Operation of Three-Phase Motors from Single-Phase Power."

Conference Paper, "Phase Converters, Their Application and Current Demand," by G Huber. Paper No. 34-CP65-356 of the IEEE Rural Electrification Committee presented 9th annual IEEE Rural Elec. Conf., San Francisco, May 24-25 1965.

"Variable-phase Polyphase from Single-Phase Supply," by John J. Vithayathil, ***Electronics,*** October 19, 1964, pp. 56-57.

Of the three, the latter is the most detailed, with vector diagrams, schematics, and component values for a given use. Briefly, the heart of the system is a center-tapped or autotransformer secondary. This is no problem for most junk boxes, as two identical nontapped units can be connected with their primaries in parallel or secondaries in series, or if both are 120-volt units, primary and secondary windings can be connected in series and the secondary series connection will become the center-tap, to give you your 220-V in and out totals. Or, you can leave the primary(ies) open and connect the 220-V single-phase power to the outer legs of the secondary(ies) to run as an autotransformer. An obvious advantage of the primary is the capability to use a 230-V motor on a 115-V service supply line, should this be desired.

The advantage of the following setup is that the output can be 120° phase-shifted between each two legs, just what you're looking for. The disadvantage is the larger number of components required.



Obviously, your motor will connect to P1, P2, and P3. If it runs backwards, reverse any ***two*** connections at your binding posts. If you are running a delta-connected motor (usually marked 208 V), you don't need the neutral connection and you may eliminate R2 and R3 from the circuit. A star-connected motor (usually marked 240 V) should have the neutral connected for best results.

Now for component values: here you will have to experiment if you don't have charts and graphs and vector algebra. The resistor across the autotransformer secondary (with cap in series) can be a variable resistor (rheostat). The reason for variability is to get all your values; switch to a fixed resistor of value determined during testing when everything is running to satisfaction. Suitable voltage and power ratings are assumed, so for your total 4-amp power draw, my guess is you'll need something under 100 ohms for each resistor, perhaps that much to start, and you may end up with under 10 ohms for one or more, and around 20 for others. AC motor-run caps will likely be 100 µF (very approximately), 400 V for cool running. Let air circulate around them, but enclose everything electrically "hot" for safety. Don't use starting caps as they are usually too underrated for constant use; use cap-run caps.

By the way, ordinary electrolytic DC capacitors can function fairly well by placing two identical DC-caps back-to-back where each capacitor in the schematic is called for. They will malfunction in short order if not back-to-back (positive to positive or negative to negative). But if you want a permanent installation, go for cap-run AC units. If underrated and/or overheated, caps may explode, so be sure to enclose the works in a suitable air-circulating box.

I realize that all this is quite likely more grief than you feel is worth the hassle, but the circuit, with proper components, gives a beautiful, equal voltage per phase under power. It is also fairly broad in retaining that equality over variations in load on your motor. By the way, you can substitute four reactances (such as simple saturable reactors) for the caps. I just happen to have three 50-amp jobs (very, very weighty!) that I used way back then instead of caps. Beautiful, because all I had to do was vary the reactor control voltages and measure the voltages across each phase leg to eventually come out with the correct resistor values. You can make these from ordinary power transformers with 120-volt primaries in series and the secondaries bucking-connected as control windings. The efficiency of ordinary transformers in saturating service is very poor, however, due to their normally nonsaturating hysteresis curves.

My approach those many years ago was to make a bench-type setup using small caps and approximately 10k resistors to start, and then cut and try using a variable 1000-ohm resistor and 10-µF cap at R1/C1, if memory serves me better than usual in my late years. I was interested in a 90" variable phase-shifter at the time, but for a fixed job such as yours, you can use fixed components. I eventually ran a quarterhorse 3-phase induction motor with this setup.

The "hum" you mentioned can come from a motor trying to start but not yet running, but also can indicate

WE'RE NO. #1

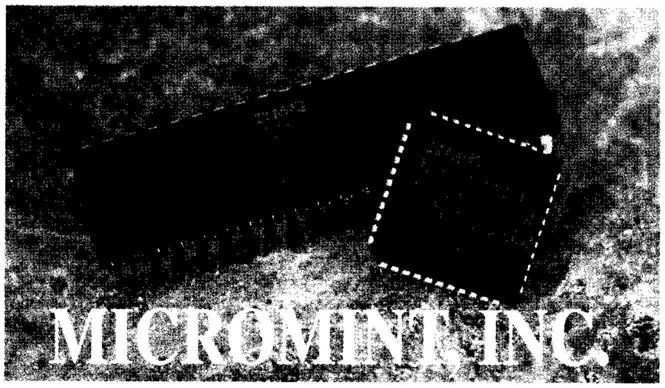
80C52-BASIC CHIPS IN VOLUME

Micromint's 80C52-BASIC chip is an upgraded replacement for the venerable Intel 8052AH-BASIC chip. Ours is designed for industrial use and operates over the entire industrial temperature range (-40°C to +85°C). Available in 40-pin DIP or PLCC..

80C52-BASIC chip **$19.00**

OEM 100 qty. **$12.00**

BASIC-52 prog. manual **$15.00**



Call (203) 871-6170 or **1-800-635-3355** to order

MICROMINT, INC. 4 PARK STREET, VERNON, CT 06066

#119

Cimetrics
TECHNOLOGY

Linking Microcontrollers. Breaking Boundaries.

The 9-Bit Solution

The Cimetrics Technology 9-Bit Solution is a complete microcontroller network (µLAN) that supports the 8051, 68HC11, 80C186EB/EC, and many other popular processors. The 9-Bit Solution takes full advantage of microprocessor modes built in to microcontollers.
The 9-Bit Solution allows simple and inexpensive development of master/slave multidrop embedded controller networks.

Link Your Product With A Cimetrics µLAN Today! 617.350.7550

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

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

#134

out-of-phase (should be 120°) in one or more of the legs, in running motors. This causes greater than necessary current draw, and can burn up a motor if you aren't careful. The system described starts by itself, runs great. No starting caps needed for only one horsepower. Besides, both resistor size and cap (or reactor) sizes vary the phase relationships—so you wouldn't want to use just starting caps alone...you'd need starting resistors as well to do the job well.

Another little gimmick I used was to wire three 15-watt light bulbs into the circuit, each across two legs of the 3-phase output. Your eyes can sense the varying light intensity quickly when you are younger, so this setup is quite rough, but for us older fellows with slower reaction times, it works better. When all three bulbs seem to be of equal brightness, you are close to the component sizes you need. In short, the bulbs and the motor are wired in parallel.

We invite you call the Circuit Cellar BBS and exchange messages and files with other Circuit Cellar readers. It is available 24 hours a day and may be reached at (203) 871-1988. Set your modem for 8 data bits, 1 stop bit, no parity, and 300, 1200, 2400, 9600, or 14.4k bps. For information on obtaining article software through the Internet, send E-mail to info@circellar.com.

ARTICLE SOFTWARE

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

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

I R S

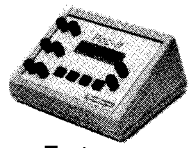
428 Very Useful 429 Moderately Useful 430 Not Useful

If you design:
Circuits, Systems, or Embedded Software

ONLY $295

Then you need:

Poc-it



Power-On **Cycler - Intermittent Tester**

The circuits, boards, and systemswe design today are complex. We are using more complex components to create these designs. Thus, we must test these components over a wide range of scenarios. For example, we have found VLSI chips, software librariesand hardwaredesignsthat **intermittently** fail to power up properly. To minimize costly re-work and embarrassing failures, we must test these designs across a large number and wide variety of power-up scenarios. Will your design power up every time?

Poc-it is designed specifically to help you test your design for just these scenarios. Featuring:

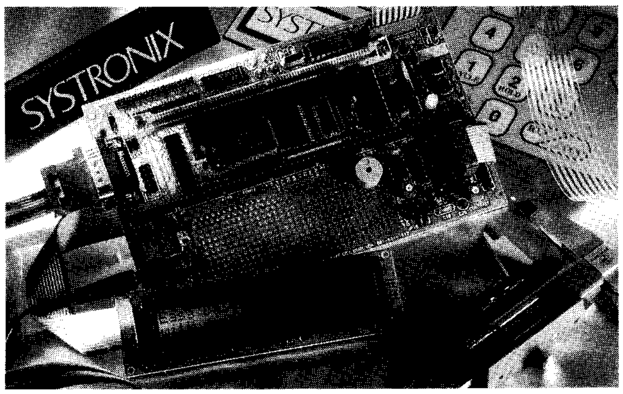
- 10 amp 120 VAC receptacle
- 10 amp relay
- Two 5 VDC high speed inputs
- One IO-30VDC sense input
- Outputs easily programmable from 0.01 **sec** to 100 min.

MICROTOOLS INC

714 Hopmeadow St., P.O. Box 624, Simsbury, CT 06070
(600) 651-6170 FAX: (203) 651-0019

#135

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



- It's a complete **8051-family** single board computer!
- One board accommodates any 40 DIP **DS5000, 40** SIMM **DS2250, 40** SIMM DS2252, or 72 SIMM DS2251, 8051 **superset** processor! Snap one out, snap another in.
- Programs via PC serial port. Program lock & encrypt.
- LCD interface, keypad decoder, RS232 serial port, 8-bit ADC, four relay driver outputs, four buffered inputs.
- Power with 5VDC regulated or 6-13 VDC unregulated
- Large prototyping area, processor pins routed to headers
- Optional enclosures, keypads, **LCDs,** everything you need
- **BCI51** Pro BASIC Compiler w/50+ Dallas keywords $399

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

#136

Necessity: the Mother of Invention

We all go through these phases in life when certain desires transcend logic. For the vast majority of us, it usually manifests itself as a craving for an unachievable goal like a V12 Ferrari or our own personal Cray computer.

Of course, the degree of eccentricity in these'phases is what separates the dreamers from the pragmatists. Dreamers are often content merely formulating fantasies while pure pragmatists often lose the entertainment value of the fantasy in an ardent quest for the goal. Somewhere in between is the person who concocts an idea and assembles the resources to achieve it. If you can understand that, you can comprehend my latest escapade.

First, I don't smoke and never have smoked. I work in a smoke-free office filled with former cigarette smokers, who could instruct religious zealots. I live in a smoke-free home and ride only in smoke-free vehicles. I hate the smell of yesterday's butts and choke when I walk into a smoky room.

Unfortunately, every once in a while I do like having a good Partagas or Royal Jamaican cigar (though I don't inhale) with a snifter of Napoleon brandy. For years, I've spent 2 months on the deck having at most a half dozen cigars and 10 months thumbing through old issues of Cigar Aficionado, pining for my next 2 months on the deck.

While not elucidating all the gory details, let me just say that during one of my recent phases I filled a lot of real estate around the house with specific-use buildings. The justification for one of these structures was that it contained an area where I could go have a leisurely cigar.

Of course, this being New England, the building had to be insulated and heated if I didn't want brandy served with an ice pick. And, if I wanted to enjoy actually being there with a cigar, then it needed certain accoutrements-a projection TV, surround-sound stereo, plush rugs, a sofa, air conditioning, and a nice lounging chair. The only problem was that the first time I sat in there and pulled out a cigar, I couldn't bring myself to light it. The place was too much like the living room. Since I wouldn't light a cigar in my living room, why would I light one here?

The only solution was to move my daydream and the cigars into the workshop. Now, instead of a stereo I have the sound of a pair of charcoal air filters simulating a jet taking off. Instead of a Mitsubishi projection TV, I watch a Fisher cast off with broken IR remote. instead of a leather easy chair, I now use a plastic lawn chair. Instead of the living room, I'm now in a grungy workshop. But, hallelujah, I can light a cigar without guilt.

Unfortunately, this has created a new problem. When I merely fantasized about cigars, I didn't have to buy or store them. Now that I might go through a box or two a year, I am very concerned about their proper care and feeding. Typical storage methods involve using a boxed humidor (nothing more than a pretty $250–$500 wooden box with a wet sponge), which is basically useless, up to a commercial walk-in humidor, which has HVAC and humidity controls. Typical prices start at about $15,000. There doesn't seem to be much in between.

It bothers me that there is no choice between something useless and something so expensive. Sounds like a project here folks. Let's see. If I take a small refrigerator, add a heating element and a fan, throw in a little saturable surface area and a water pump, screw in a couple of temperature and humidity sensors, instrument it all to an easy to use little controller like Domino (or perhaps a Blackjack telecontroller so it could phone me at the office and tell me that my Partagas were getting brittle [grin]), add some spectacularly performing PID software, and I'm in business. Of course, I could just build a room that was a little plushier, with a nice TV, a little humidity control.. .Argh!. Stay tuned.

Steve