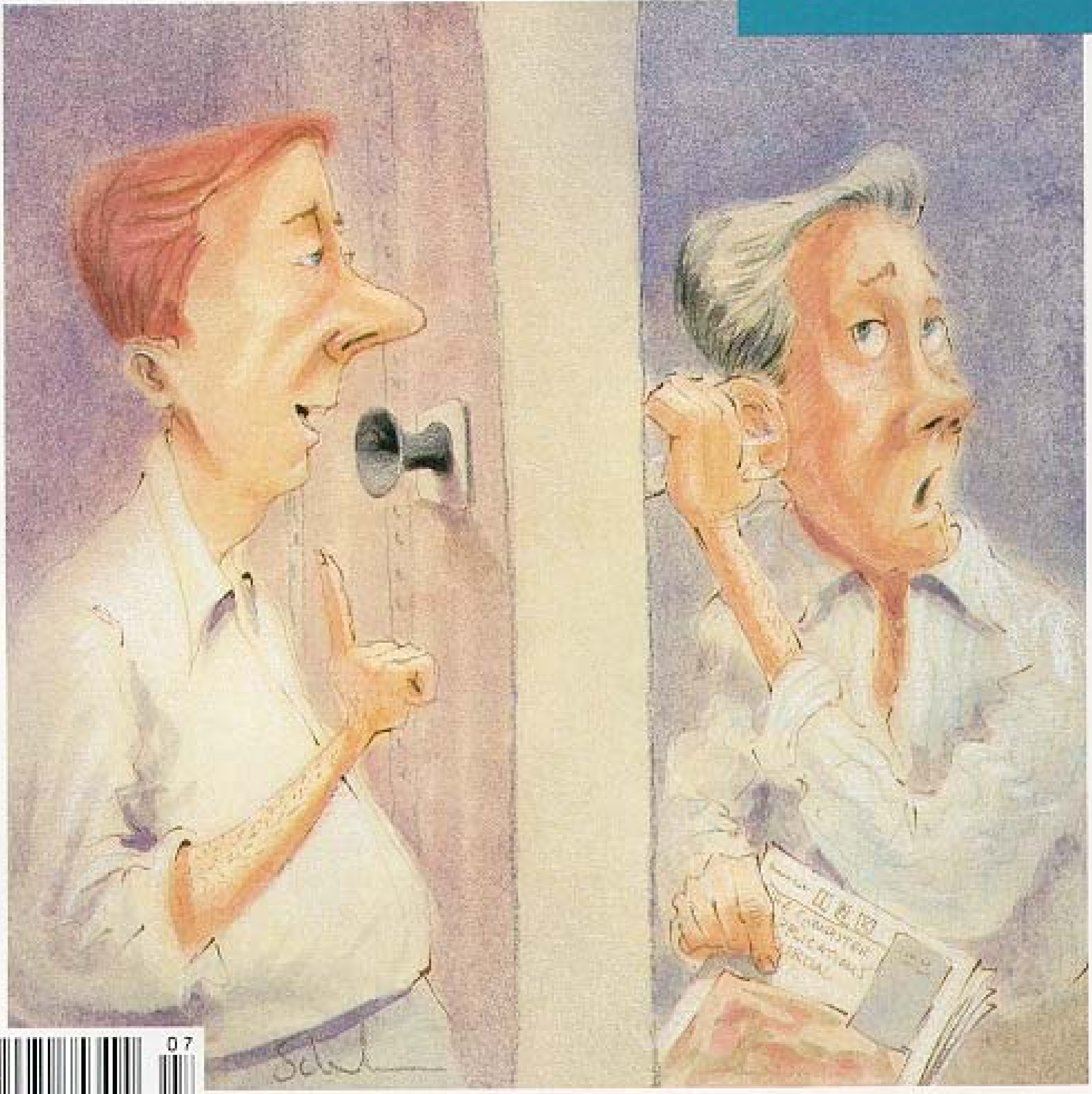
CIRCUIT CELLAR INK®

# THE COMPUTER APPLICATIONS JOURNAL

July 1994 — Issue #48

## COMMUNICATIONS



$3.95 U.S.
$4.95 Canada

# EDITOR'S INK

## Information Society

as our society grows more and more dependent on information as a commodity, we grow more and more dependent on our communications systems. After all, what good is information if it can't be used, sold, or shared? The much-ballyhooed "information superhighway" would be nowhere without the extensive telephone network already in place necessary to tie everything together. That the network is increasing in data-handling capability every day through the expanding use of fiber and microwave is a testament to this communications explosion.

While businesses have come to depend on an internal phone system to ease peer-to-peer communications, the convenience afforded by such a system, for the most part, hasn't made the jump to the home. In our first article, we take a look at a simple in-home PBX that uses regular telephones for performing intercom-like functions. The same project can be used as a base for future expansion to extend the capabilities to the outside world.

Next, garbled data is as good as no data at all, so any information transfer over imperfect communications lines must include some kind of error detection or correction. The Golay error detection and correction code was first theorized back in 1949, but it's finding uses in many of today's applications.

While perfect data is important, often that data must also be protected from prying eyes. Data encryption is a hot topic of debate these days, with the Federal Government getting involved in the latest round. Our third article looks at some of the current issues to give you an idea of what some of your elected officials are doing.

In our final feature article, we extend the usefulness of the popular and low-cost RS-485 network to accommodate more than 32 devices, longer cable runs, and more flexible network topologies.

In this month's columns, Ed bows to popular demand by starting a series of articles on how to do protected mode programming on the '386SX. Jeff finishes up his exoskeleton project with some support software. Tom has finally recovered from last month's journey to LA with a look at an emerging standard that promises to ease the task of wireless communication between palmtop and handheld computing devices. Finally, John continues his BIOnet coverage by writing some reusable software.

[illegible]

CIRCUIT CELLAR INK®

**THE COMPUTER APPLICATIONS JOURNAL**

FOUNDER/EDITORIAL DIRECTOR
Steve Ciarcia

EDITOR-IN-CHIEF
Ken Davidson

TECHNICAL EDITOR
Michael Swartzendruber

ASSOCIATE EDITOR
Rob Rojas

ENGINEERING STAFF
Jeff Bachiochi & Ed Nisley

WEST COAST EDITOR
Tom Cantrell

CONTRIBUTING EDITORS
John Dybowski & Russ Reiss

NEW PRODUCTS EDITOR
Harv Weiner

ART DIRECTOR
Lisa Ferry

GRAPHIC ARTIST
Joseph Quinlan

CONTRIBUTORS:
Jon Elson
Tim McDonough
Frank Kuechmann
Pellervo Kaskinen

Cover Illustration by Bob Schuchman
PRINTED IN THE UNITED STATES

PUBLISHER
Daniel Rodrigues

PUBLISHER'S ASSISTANT
Sue Hodge

CIRCULATION COORDINATOR
Rose Mansella

CIRCULATION ASSISTANT
Barbara Maleski

CIRCULATION CONSULTANT
Gregory Spitzfaden

BUSINESS MANAGER
Jeannette Walters

ADVERTISING COORDINATOR
Dan Gorsky

CIRCUIT CELLAR INK, THE COMPUTER APPLICATIONS JOURNAL (ISSN 0896-8985) is published monthly by Circuit Cellar Incorporated, 4 Park Street, Suite 20, Vernon, CT 06066 (203) 875-2751. Second class postage paid at Vernon, CT and additional offices. One-year (12 issues) subscription rate U.S.A. and possessions $21.95, Canada/Mexico $31.95, all other countries $49.95. All subscription orders payable in U.S. funds only, via international postal money order or check drawn on U.S. bank. Direct subscription orders and subscription related questions to The Computer Applications Journal Subscriptions, P.O. Box 7694, Riverton, NJ 08077 or call (609) 766.0409.
POSTMASTER: Please send address changes to The Computer Applications Journal, Circulation Dept., P.O. Box 7694, Riverton, NJ 06077

**HAJAR** ASSOCIATES NATIONAL ADVERTISING REPRESENTATIVES

NORTHEAST
Debra Andersen
(617) 769-8950
Fax: (617) 769-8982

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

# INSIDE ISSUE 48

# READER'S INK

## AVMux Clarifications

Thank you, Steve, for the plug you gave Maxim's video products in your article "Control Your Audio/ Video Connections with the AVMux" (April 1994). The circuits are extremely interesting and I for one am anxious to build it as are some other engineers (who are not employed by Maxim).

For completeness, I should point out a minor error in the schematic of Figure 5. I know you may have been led astray by our MAX456 data sheet, so I do place the blame on Maxim and not you. In Figure 5, the inputs should be terminated with 75-ohm resistors since the MAX456 does not contain an input termination resistor. Additionally, the output lines may confuse some people. The 75-ohm resistors to ground at outputs 1-8 are meant to represent the total load. If the builder's equipment contains a 75-ohm termination, the resistor(s) should be eliminated. They were included in the data sheet schematic only to show that the 75-ohm cable must be terminated at the load end.

Another point is that the note under Figure 5 indicates the use of "shielded cable." There are many types of shielded cable, but in video circuits you want to use a 75-ohm coaxial cable such as RG59/U.

For your readers' further information, Maxim has just introduced the MAX458 and MAX459 crosspoint video switches. These are 8 x 4 (8 in/4 out) devices that contain amplifiers capable of driving a 50-/75-ohm back terminated cable. The MAX459 is a gain-of-2 device to enable the user to achieve a total throughput gain of 1 when driving a back-terminated cable. The MAX458 gain is 1 and is particularly useful when building larger arrays.

Joe Stern
Applications Engineer
Maxim Integrated Products
Sunnyvale, Calif.

## Last Word on Program Labels

Jeff Mills's program-labeling technique ("Reader's INK," May 1994) leaves a lot to be desired. His examples may be useful for simple 20-line programs, but labels should tell me what the code is doing. For example, I prefer calling a routine labeled CRLF instead of DO_1 when it comes to putting out simple carriage return and linefeed bytes. When I look at CALL CRLF in the midst of 5000 lines of code, I have a good idea of what the code should do. CALL DO_1 tells me nothing. Also, when the CRLF routine finishes, I'd much rather have it jump to END_CRLF than jump to WHILE_1.

Jon Titus
Milford, Mass.

## Sony Info, Please

I am an avid reader of Circuit Cellar INK. I also happen to be into computer graphics and use Sony camcorders supporting Control-L and their Umatics which support RS-422 ports for data communication. I am trying to make an edit controller which can be interfaced to my Amiga to record my animations.

Well, the problem is that I cannot find any data on these protocols (i.e., what are the commands to which the recorders respond, their format, etc.). Locating this type of data in Pakistan is really difficult. I was hoping you could help me in this issue. My Email address is brain!786bbs.uucp!mudassir@uunet.uu.net.

Mudassir **Farhat** Khan

*Can anyone help out this gentleman! If you have information but no Internet Email access, send it to us and we'll be sure to forward it to him.*

## Contacting Circuit Cellar

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

Mail: Letters to the Editor may be sent to: Editor, The Computer Applications Journal, 4 Park St., Vernon, CT 06066.
Phone: Direct all subscription inquiries to (609) 786-0409. Contact our editorial offices at (203) 875-2199.
Fax: All faxes may be sent to (203) 872-2204.
BBS: All of our editors and regular authors frequent the Circuit Cellar BBS and are available to answer questions. Call (203) 871-1988 with your modem (300–14.4k bps, 8N1).
Internet: Electronic mail may also be sent to our editors and regular authors via the Internet. To determine a particular person's Internet address, use their name as it appears in the masthead or by-line, insert a period between their first and last names, and append "@circellar.com" to the end. For example, to send Internet Email to Jeff Bachiochi, address it to jeff.bachiochi@circellar.com. For more information, send Email to info@circellar.com.

# NEW PRODUCT NEWS

Edited by Harv Weiner

## PC-LESS MICROCONTROLLER NETWORK

Cimetrics Technology has announced two additions to its 9-bit Solution line, an array of software/hardware packages which enable designers of embedded systems to network their embedded Intel, Motorola, or Zilog microcontrollers. By linking up to 250 CPUs using each processor's built-in serial port, the 9-bit Solution microcontroller networks (or µLANs) provide a cost-competitive, development-sensitive system for adding monitoring, diagnostics, control, or data acquisition capabilities to CPU-based products or subsystems.

The Microcontroller Master Software for Intel 80C **186** and Motorola 68HC 11 enables the system designer to create a network composed entirely of embedded microcontrollers; a PC need not be a component of the completed network. This type of "PC-less" embedded controller network is particularly well-suited to use in distributed control and data acquisition systems where polling must occur automatically and a user interface is not required. C language source code is provided for easy modification and maintainability.

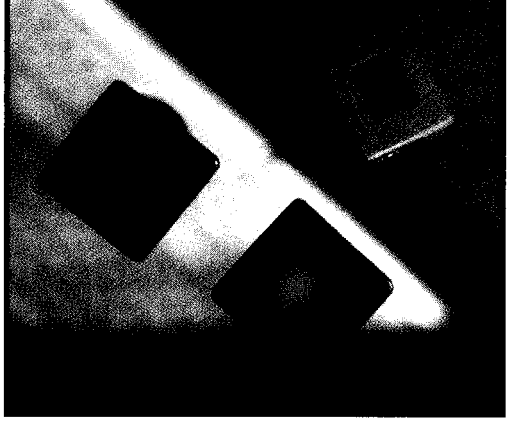
The NBS-10 Device Driver for Windows NT is designed for RS-485 and RS-422 applications. This driver, which enables PCs to participate in a microcontroller network, offers programmable 5–9-bit asynchronous character format at 288,000 bps. The multitasking assets of Windows NT and the above hardware and software provide an ideal platform for the designer who wishes to analyze and process data concurrent with data acquisition.

The Microcontroller Master sells for $1500 as stand-alone software and the NBS-10 device drive sells for $500 plus $10 per CPU. A complete 9-bit Solution Package, consisting of the Network Software, Network Monitor software, and two NBS-10 interface cards, is available for $1995.

Cimetrics Technology
120 West State St. • Ithaca, NY 14850
(607) 273-5715 • Fax: (607) 273-5712 **#500**

## SQUARE **LEDs**

A way to efficiently mount contemporary-styled square LEDs in inexpensive round holes has been provided in a new style of front-panel LED indicators from Lumex. The SSI-LXH55 series fits into standard $^{11}/_{32}$" round holes, while the outside dimensions are a perfectly square 5 x 5 mm (0.2" x 0.2") for the lighted area and 10 x 10 mm (0.4" x 0.4") for the surrounding black bezel.

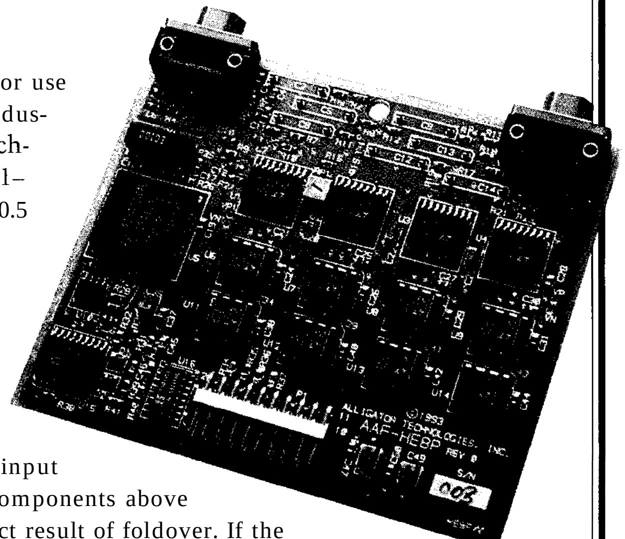


Five color choices include blue, green, yellow, amber, and red, with bicolors available as specials. Light intensity varies from 14 to 50 mcd depending on color. Other options include 5- and 12-volt designs with an integral current-limiting resistor chip and a 2.5-volt style without current protection. Terminations are 0.020" square, 1" long tin-dipped pins or 26 AWG insulated leads of any length. Prices range from $0.25 to $0.75 depending on model.

Lumex **Opto/Components,** Inc.
292 E. **Hellen** Rd. • Palatine, IL 60067
(708) 359-2790 • Fax: (800) 944-2790 **#501**

# NEW PRODUCT NEWS

## ANTIALIASING FILTER



A rugged, RS-232 programmable, low-pass filter unit targeted for use in front of any A/D converter in avionics, automotive, and other industrial data acquisition systems has been introduced by Alligator Technologies. The AAF-HESP is an eight-channel board that features a 1–50-kHz software-selectable cutoff frequency range, DC accuracy, 0.5 degree phase matching between channel, and a five-pole Butterworth filter with a maximally flat passband and low phase lag. Capable of withstanding high vibrations and extreme temperatures, the AAF-HE8P filter is ideal for such applications as airborne instrumentation and control, on-ground vehicle or engine testing, shock and vibration measurement, ultrasonics, sonar and acoustic analysis, and robotics.

Designed to ensure data integrity, the AAF-HESP allows valid input signals to pass while removing undesired signals with frequency components above one-half the sampling rate of the A/D converter. (Aliasing is a direct result of foldover. If the sampling rate is set at 200 Hz, then frequency components at or higher than 100 Hz will fold over into the lower frequency spectrum and appear as aliased events that may not be distinguishable from valid sampled data. Using an antialiasing filter with sharp attenuation eliminates the need to oversample the input signal.)

The AAF-HESP filter is programmable through an RS-232 port, allowing users to control the cutoff frequency of up to 16 units with a single RS-232 link. The frequency setting of the filter remains through power on-off cycles until changed by new programming. One three-byte communications sequence is sent at standard protocol and controls the cutoff frequency for all eight channels. Complete verification of operation is returned to the host to aid in system diagnosis.

Each AAF-HESP has one 20-pin power and serial interface connector as well as two 9-pin I/O connectors. The AAF-HESP Antialiasing Filter configured with eight filtered channels sells for $3950. Substantial discounts are available for OEMs, system integrators, and multiple end users.

Alligator Technologies
2900 Bristol St., Ste. E-101 • Costa Mesa, CA 92626-7906 • (714) 850-9984 • Fax: (714) 850-9987 **#502**

## EMBEDDED CONTROL HANDBOOK

Microchip Technology has announced the availability of its latest Embedded Control Handbook, a 982-page technical reference tool created for systems and applications designers using PIC S-bit microcontrollers, and specialty nonvolatile memory products.

The handbook included more than 50 application notes and pieces of software written for specific embedded control applications such as "Servo Control of a DC Brush Motor," "Logic-powered Serial EEPROMs," and "LCD Controllers." Additional highlights include schematic and timing diagrams, math routines, and associated figures for comprehensive application support.

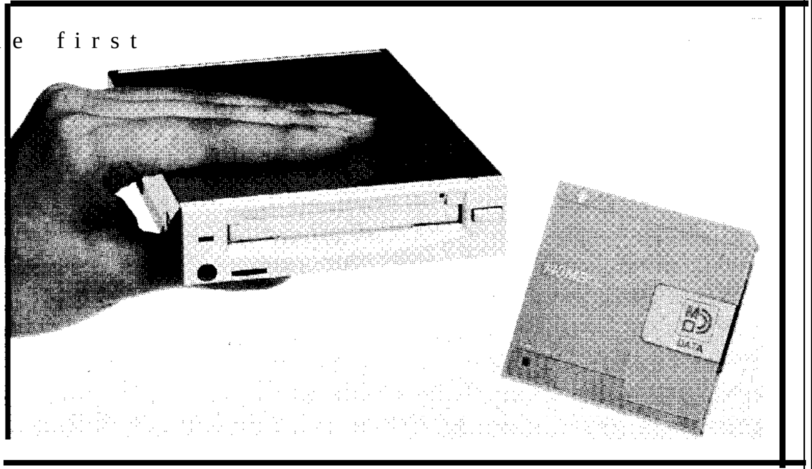
Microchip Technology, Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
(602) **786-7200**
Fax: (602) 899-9210

**#503**

# NEW PRODUCT NEWS

## MD DATA DRIVE AND MEDIA

Sony has introduced the first **MD DATA drive (MDM 111)**, a rewritable storage device based on Sony's MiniDisc personal audio technology. When combined with rewritable MD DATA media (MMD-140), the format provides exceptional flexibility for data management, storage, and distribution.



The MD DATA drive and media provides 140 MB of storage on a magneto-optical disc measuring 2.5" in diameter. The MD DATA drive accepts rewritable, read-only, and hybrid (with both read-only and rewritable sections) media. With the ability to handle various types of digital information, MD DATA is ideal for applications such as personal backup, interactive software, electronic publishing, and easy transportation of large files.

In contrast to conventional magneto-optical drives, MD DATA drives rewrite discs in a single pass. Size, cost, and power consumption are thereby reduced, making MD DATA ideal for portable applications. Discs can also be rewritten with virtually no loss of data integrity.

A new universal file system allows MD DATA discs to be used interchangeably between computers with the currently most popular operating systems. Besides computer applications, MD DATA drives can also play prerecorded music titles and user-recorded audio MiniDiscs.

Sony Electronics
3300 Zanker Rd. • San Jose, CA 95134 • (800) 352-7669

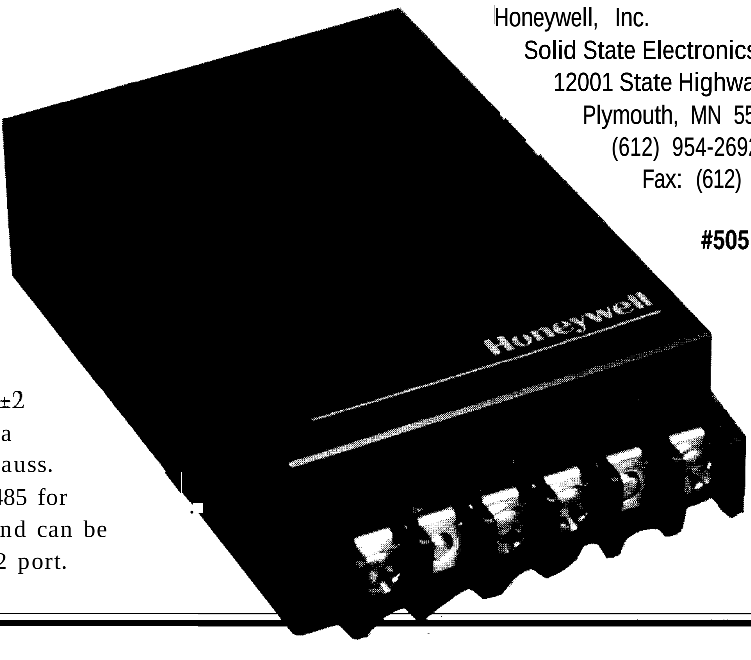
**#504**

## DIGITAL MAGNETOMETER

**A smart magnetometer** has been introduced by the Solid State Electronics Center of Honeywell. The unit has applications in compassing, attitude positioning, traffic and vehicle detection, remote vehicle monitoring (yaw/roll/pitch), security systems, and process control.

The magnetometer uses a microcontroller to produce a three-axis digital output device that features high sensitivity, low power consumption, ease of use, and affordability. The smart digital magnetometer has a measurement range of ±2 gauss with **1%** accuracy and a sensitivity level of 0.5 milligauss. The device interfaces to RS-485 for multiport bus transmission and can be easily interfaced to an RS-232 port.

The unit is based on a magnetoresistive thin film permalloy process. The on-board microcontroller handles the magnetic sensing, first-order temperature correction, and all output communications.



Honeywell, Inc.
Solid State Electronics Center
12001 State Highway 55
Plymouth, MN 55441
(612) 954-2692
Fax: (612) 954-2051

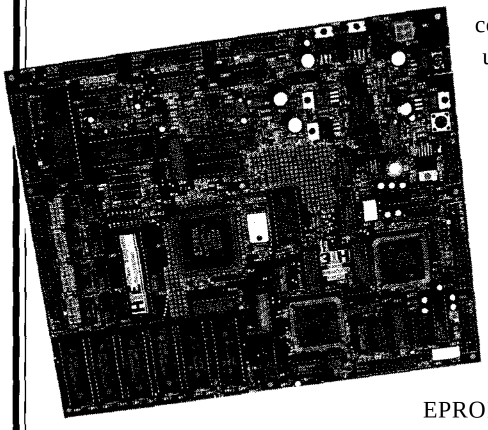
**#505**

# NEW PRODUCT NEWS

## SINGLE-BOARD COMPUTER

HiTech Equipment Corporation has announced a new single-board computer based on the Intel 80C188XL processor. The **188SBC** provides 12- or 16-bit A/D and D/A converters, serial ports, a parallel port, opto rack interface, LCD and keyboard interfaces, real-time clock, and a PC/ 104 expansion bus.

A unique feature of the board is a Field Programmable Gate Array, available for user customization. When configured by the 80C188, the gate array can take on the personality of any digital circuit the user may require for a special interface or other application-specific circuit. The Xilinx 3000 series FPGA socket accepts the XC3030 through XC3090 parts containing from 100 to 320 Configurable Logic Blocks for a wide range of flexibility. A breadboard area on the board allows prototyping of level converter circuits for the FPGA.



The 188SBC is complemented by up to 5 12K bytes of battery-backed, fully static RAM. Two additional sockets are provided for code, compatible with 1- and 2-megabit EPROMs or flash memory. On-board flash programming is provided.

Two serial ports are available, each selectable as either RS-232 or RS-485. Data rates are programmable from 50 to 38.41~ bps. The bidirectional parallel port is configured as a Centronics-compatible interface.

Other digital interfaces include a direct interface to character and graphics LCDs. The interface is compatible with the Hitachi 44780, Toshiba T6964, SMOS 1330, and other controllers. One 82C55 programmable peripheral interface chip provides a direct interface to optical isolation/relay racks similar to Opto-22 and Grayhill products. LED indication of output and input states is available on all 24 control lines. A second 82C55 provides an interface to additional external I/O such as a keyboard matrix.

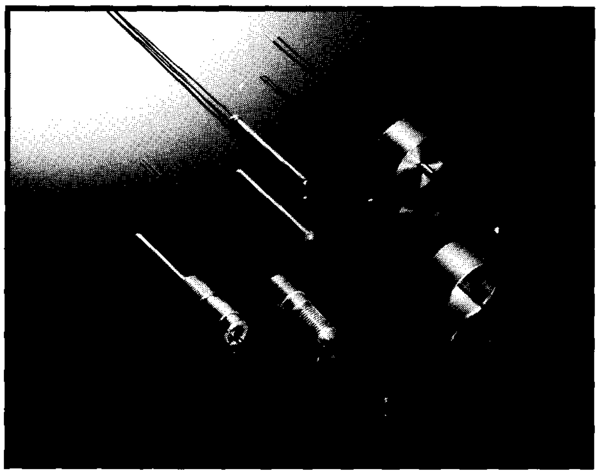
The analog portion of the board consists of a 16-channel, 12-bit A/D converter with a 25-µs conversion time. A common low-pass antialiasing filter is provided and input voltage levels may be ±5 V or O-10 V. The converter is specified with a ±1.5 LSB accuracy, and an optional 16-bit converter is available. The D/A converter also has 12-bit resolution. A SMP-08 sample-and-hold multiplexer creates eight D/A output channels for the converter.

Power to the system is provided by an on-board switching regulator. This regulator supplies quiet power to the analog circuits via isolated power and ground layers on the circuit board. Input to the regulator can range from 12 V to 20 V AC or DC and power consumption is typically 350 milliamps, fully loaded. A nonmaskable Power-Fail interrupt is provided. The 188SBC price starts at $299 and tops out at under $750 for a fully loaded board.

**HiTech** Equipment Corp.
9400 Activity Rd. • San Diego, CA 92126
(619) 566-I 892 • Fax: (619) 530-I 458

**#506**

## FLUID-LEVEL SENSORS



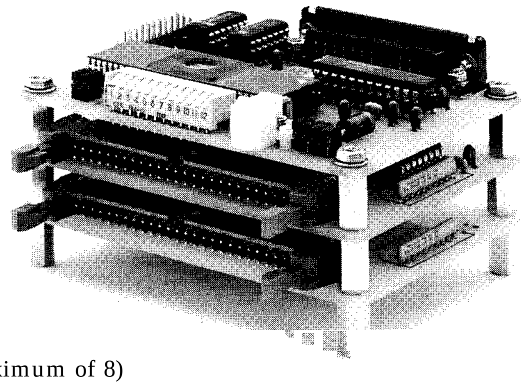
A new line of metal-encapsulated **fluid-level sensors** for monitoring temperature has been announced by the Special Products Division of Siemens Components. Housed in a noncorroding CrNiMo steel housing, the sensors can be used in water, oil, gasoline, and detergent solutions. As such, they are ideal for chemical processing and petroleum refining applications. The fluid-level sensors are available in a choice of two temperature ranges: -25°C to +50°C and +10°C to +90°C.

Siemens Components, Inc.
186 Wood Ave. South
**Iselin,** NJ 08830 • (908) 603-5919

**#507**

# NEW PRODUCT NEWS

## DIGITAL INTERFACE DEVICE



MillRace Software has announced a new concept that effectively locates a computer interface card in a custom hardware design. The compact design minimizes cables and increases I/O bits up to 320 lines. A high level of I/O control is available from any computer by means of a standard RS-232 connection.

The **I/O** Stack provides a computer interface with a minimum of hardware and software design. The CPU and functional boards are stacked one on top of another to provide a compact unit to be mounted inside the user's application hardware. The I/O Stack is ideal for rapid prototyping of a design or for use in one-of-a-kind applications.

It consists of a microprocessor board and one or more (up to a maximum of 8) TTL I/O boards. The microprocessor board is the stack controller and is interfaced to the user's computer through an RS-232 interface. Data rates from 150 bps to 19.2 kbps can be accommodated.

Each TTL I/O board contains 5 bytes (40 bits) of I/O lines. Each byte is either all input or output, so a TTL I/O board may contain any combination of input or output byte configurations. Input or output can be at the bit, byte, or word level. Word I/O can optionally use a 2-wire handshaking method to pace data flow.

The I/O Stack is ready to run-there is no embedded code to write. Digital control is accomplished with a simple high-level ASCII command set. Numeric parameters can be specified in decimal, hex, octal, or binary formats. The I/O Stack provides standard TTL interfacing; only 5-volt DC power is required. The I/O Stack microprocessor board sells for $495 and each TTL I/O board, in any configuration, sells for $295.

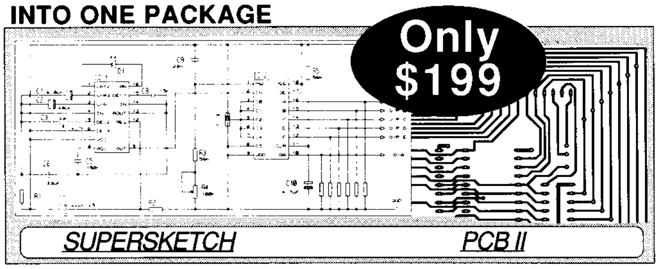
**MillRace** Software • 828 Manhattan Beach Blvd. • Manhattan Beach, CA 90266 • (310) 546-6737 **#508**

**CADPAC II**

TWO PROGRAMS FOR ONE LOW PRICE!!

**SUPFRSKETCH & PCB II:** INTEGRATED **INTO ONE PACKAGE**

**Only $199**



PCB II & SUPERSKETCH features:

* MOUSE DRIVEN *SUPPORTS CGA, EGA, VGA & SVGA, • OUTPUT TO 9 & 24 PIN PRINTERS, HP LASERJET& HPGL PLOTTERS • OUTPUT TO DTP PACKAGES • * PCB II ALSO HAS GERBER OUTPUT & VIEWING. *

**THF EASIEST TO USE CAD AVAILABLE**

R4 **SYSTEMS Inc.**
1111 Davis Drive, Suite 30-332
Newmarket, Ontario **L3Y 7V1**
**(905)** 898-0665
**fax** (605) 898-0683

*Free DEMO Package Write or Call Today*

**ALL PRICES ARE IN US FUNDS, PLEASE INCLUDE $7 S/H**

**Download DEMO from BBS at (905) 898-0508 (9600/8/N/1)**

**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 compatibile*
- *Compatible with all 8051 variants*
- ***BXC51 $ 295.***

508-369-9556
FAX 508-369-9549

Binary Technology, Inc.
P.O. Box 541 • Carlisle, MA 01741

#104

# The Personal PBX

While small-scale key system units (KSUs) are available commercially for the home, they usually have extra features and require special telephones. Instead, put your own system together for a fraction of the cost.

FEATURE ARTICLE

Richard Newman

Many years ago, I wanted a telephone system in my home that would let all the devices that use a telephone line share the two I had. Looking at commercial phone systems, I found they were all too expensive and had features meant for a business environment rather than the home. I decided to build one, and the first version worked great, using five relays for every phone. The version of the project I'm presenting here came along about 10 years later and uses only one relay per phone.

The small switching system, or Personal PBX, presented here is actually the prototype of a larger system that is used commercially in the private operator service industry to provide dial tone to hotels, dorms, and institutional environments.

In this article, I will describe how to provide basic telephone service to single-line telephones (called "2500 sets," which are served by POTS—Plain Old Telephone Service). The project will provide dial tone, receive DTMF [Dual Tone, MultiFrequency, often called Touch Tone) signals, ring the phones, and set up conversations between two or more sets.

I'll also leave open the possibility of adding more features, an interface to the public switched network, and an interface to a PC serial port for remote control and voice response.

## TELEPHONE BASICS

Your telephones at home connect to the telephone company's central office (CO) by way of two wires called the ***wire pair,*** with one wire called ***tip*** and the other ***ring.*** Curiously, these

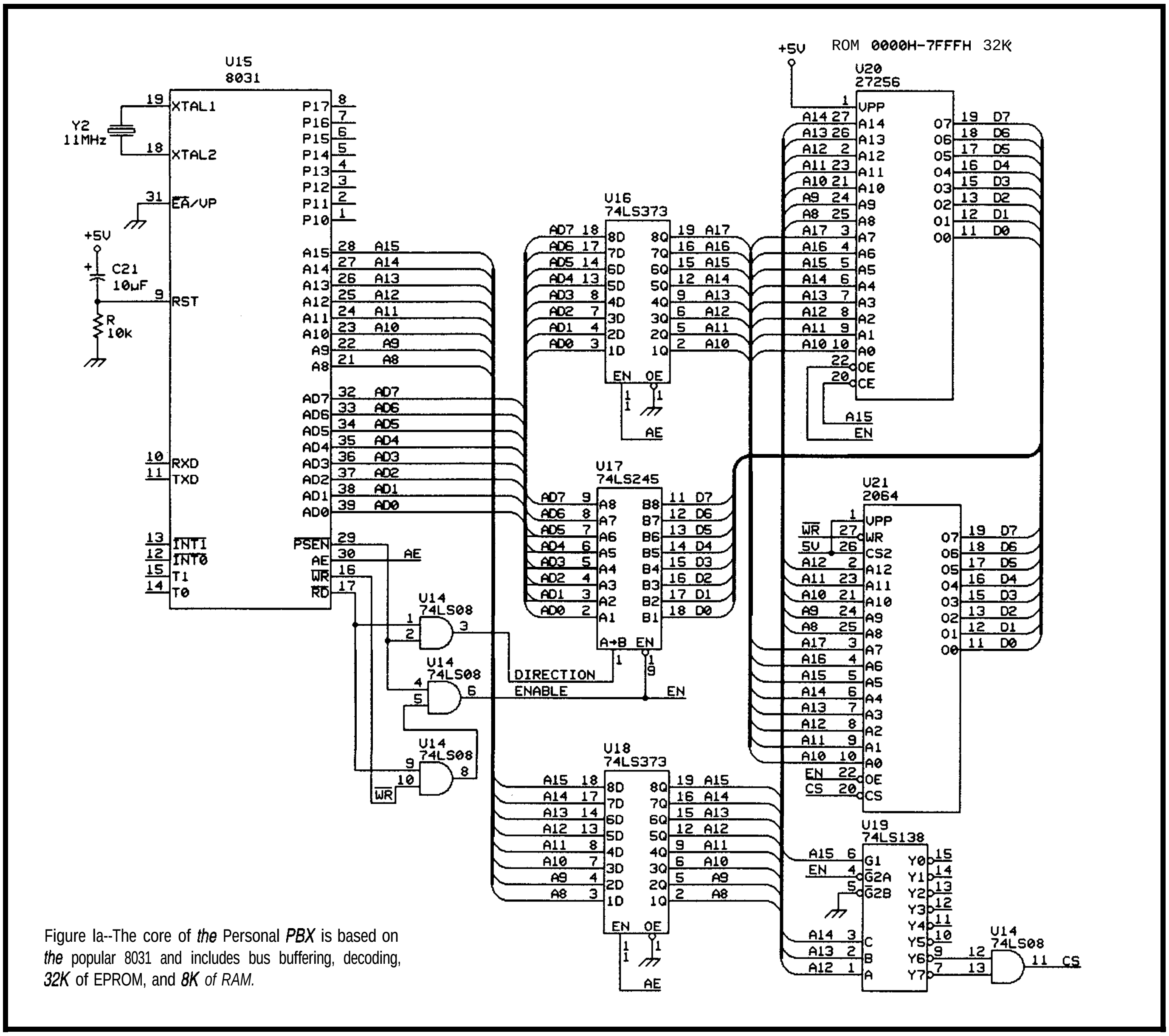


Figure la--The core of *the* Personal *PBX* is based on *the* popular 8031 and includes bus buffering, decoding, *32K* of EPROM, and *8K of RAM.*

names date back to the time when human switchboard operators manually connected calls with patch cords. Each cord had a jack similar to that found on today's headphone cords, and that jack had a tip and a ring.

When the phone is off-hook, the wires carry about 20 milliamps of current from the CO through the telephone set and back to the CO. Tip is closest to earth ground and ring, when measured from the telephone end, has about -48 volts DC on it.

When the phone is on-hook, no current flows. When someone calls your house, the CO places 90-130 volts AC on your wire pair in 1–3-second bursts to make the ringer in your phone sound. Lifting the receiver allows current to flow and a conversation can take place.

Because of the relatively high voltages used by the telephone network, special interfaces must be used to control the telephone lines and set up conversations between them.

The system I present here-the Personal PBX-does just that: illustrates interfaces for connecting telephone equipment to a controlling microprocessor.

## THE HARDWARE

There are six major hardware sections to the Personal PBX: CPU, power supply, subscriber line interface circuit (SLIC), ring generator, common control, and switch matrix.

## CPU

The project is based on a Signetics/Philips 805 1 microprocessor with 32K of EPROM and 8K of static battery-backed RAM (see Figure la). The EPROM contains the core code that controls the hardware, and the RAM contains the "translation" information that specifies the telephone number of each telephone set, the "class of service" that specifies what the phone can do, and the "class of restriction," which specifies where the phone can call (if it could call the real world, anyway).

Although I show interfaces for only one telephone set in the schematic, you can add as many interfaces as you like for as many phones as you

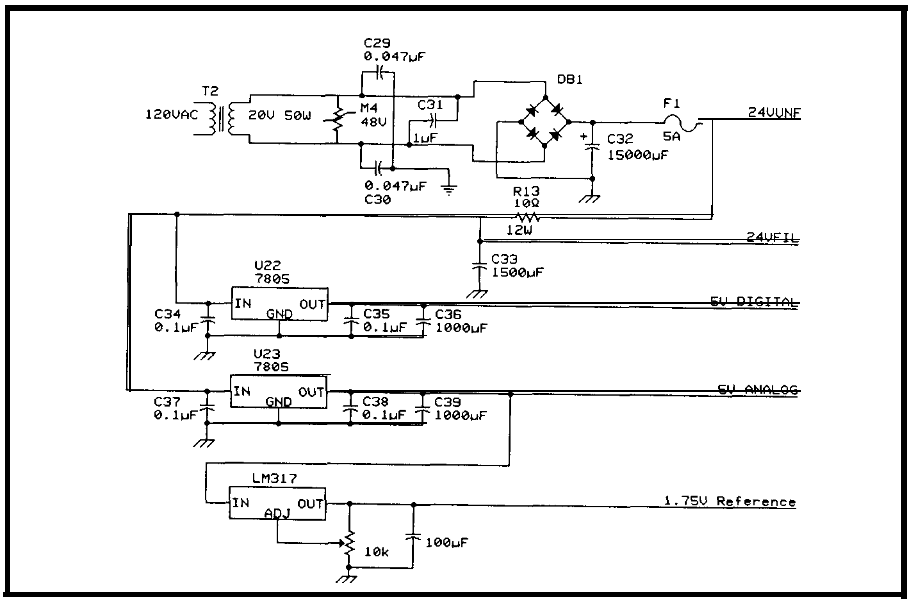


Figure 1b—*The power supply is large enough to service* 32 phones and *all the control circuitry.*

need [as long as you don't run out of I/O space or CPU time).

## POWER SUPPLY

The power supply for the system is large enough to service 32 phones and all of the common control providing service for them (Figure lb). It consists of a transformer and rectifier that provides 24-40 volts unfiltered DC, a filter arrangement that provides 24-40 volts of filtered DC (called the "talk supply"), a positive 5-volt regulator (or two) that powers all of the digital logic, and another 5-volt regulator that powers all of the analog parts including the switch matrix and common controls. A variable floating regulator provides a virtual ground for all of the op-amps and SLICs that actually sits at about 1.75 volts.

The analog portions of the system require very quiet DC supplies or the noise will be coupled into the conversation. Even though people talking on the telephone sets cannot hear the noise, the DTMF receivers can, and a noisy power supply will increase the signal detection time considerably. To this end, I provide a separate analog 5-volt supply that powers anything relating to audio, including op-amps, dial tone generators, DTMF receivers, and the virtual ground regulator.

The virtual ground is necessary because I am using a single supply for all of the audio paths between the system's subcomponents. Without a virtual ground sitting somewhere between real ground and the supply voltage, the AC waveforms would go below ground on the negative swing of the signal, causing some components to misbehave. Since all of the waveforms are referenced to the 1.75volt virtual ground, they end up swinging from about 0.25 V to about 3.25 V.

The 24 volts filtered DC is used to provide power to all of the telephone sets, which need 2030 mA in the off-hook condition. A filtered supply is required to prevent the AC line hum from being coupled into the conversation. I chose a filtering arrangement with a power resistor because it is less costly than a large inductive filter even though it dissipates lots of heat.

The unfiltered 24 V is used to power anything that is noisy, including the relays and the ring generator. When the ring generator is used, a large DC reserve from the filter cap is necessary to create a field of sufficient strength in the ring generator coil to make an AC waveform ring the bell in a telephone set.

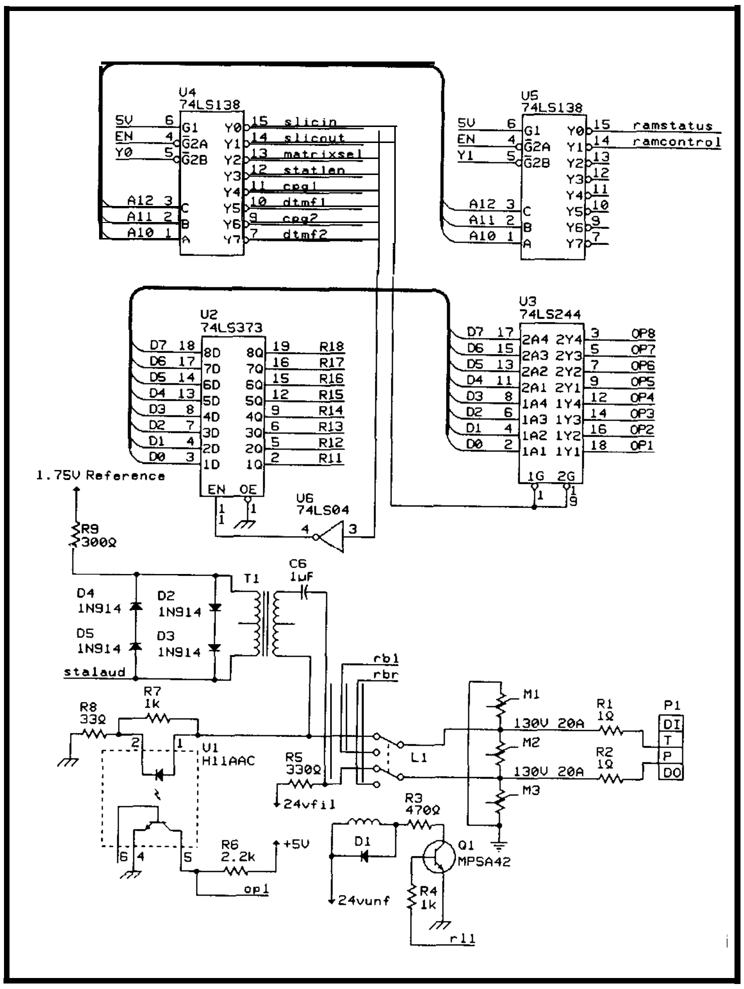


Figure 1c—*It is the responsibility of the SLIC to electrically isolate the high-voltage telephone loop from the low-voltage switching matrix,*

## SUBSCRIBER LINE INTERFACE CIRCUIT

It is the responsibility of the SLIC (Figure lc) to electrically isolate the high-voltage telephone loop that extends from the telephone equipment to the telephone set from the low-voltage and fragile switching matrix. Additional duties include indicating when the telephone is off-hook, and switching high voltage onto the line to ring the telephone set.

mA
and may extend from several hundred to several thousand feet. It is a two-wire balanced line, that is, you may not bridge the wires in the field to share a common ground between two telephone sets.

The SLIC I present here provides a nonstandard interface: the loop current may vary and the loop impedance does not reflect 600-900 ohms which is desirable in the telephone industry. However, this SLIC is inexpensive and is very workable for the Personal PBX.

The line current to the telephone set is provided by two 330-ohm ½-watt resistors: one to ground and the other to the 24-volt line. These two resistors provide subscriber loop tip and ring.

Tip and ring are then bridged onto the primary of a 600-ohm audio matching transformer through a 10-µF nonpolarized capacitor to block the DC loop current. When an audio signal from an off-hook telephone set appears at tip and ring, a facsimile of it will also appear at the secondary of the audio transformer.

One terminal of the audio transformer is connected to the analog virtual ground, so any waveform coupled to the secondary will be riding on this 1.75volt DC carrier.

On the secondary are two sets of 1N914 diodes connected so the AC waveform never exceeds about 3 volts peak-to-peak. This provides protection after the operation of the ring relay, which may otherwise couple a high-voltage spike onto the secondary and destroy the switch matrix.

When the telephone sets user lifts the receiver, the telephone allows current to flow, which in turn turns on the optocoupler in the SLIC to inform the CPU that the phone is off-hook.

When the CPU needs to ring the telephone, it operates the associated ringing relay on the SLIC and activates the common ring generator, which generates 90 VAC at 20 Hz. After providing one ring burst, the CPU releases the ring relay, returning the telephone to the audio coupling and current detection circuitry of the SLIC.

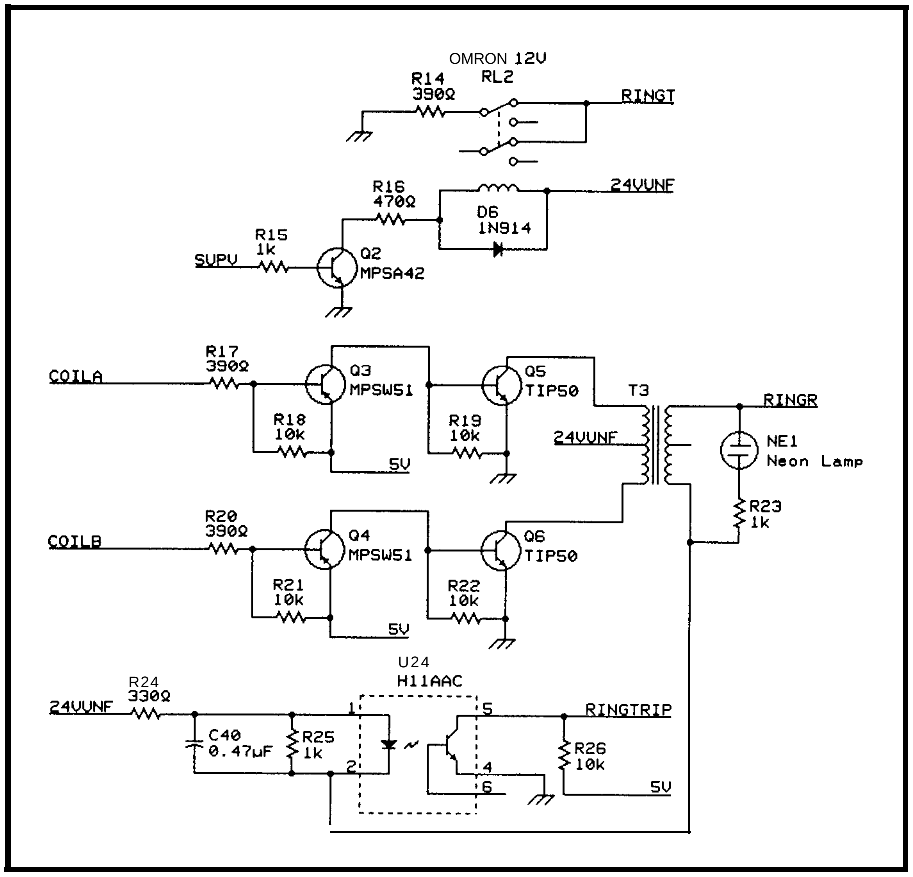
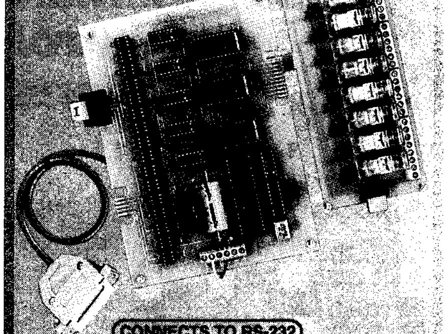


Figure 1d—The ring generator circuit converts 24 volts to 90 VAC at 20 Hz to ring fhe telephone sets.

**RELAY INTERFACE**

PROVIDES SOFTWARE CONTROL OF RELAYS

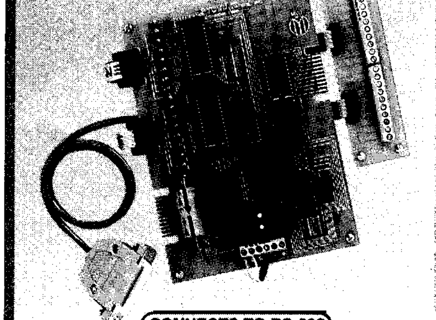


CONNECTS TO RS-232

**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) . . . . . . . . . $99.95
Input on/off status of relays, switches, HVAC equipment, security devices, smoke detectors, and other devices.
**STA-8D** TOUCH TONE INTERFACE'................ **$** 134.90
Allows callers to select control functions from any phone.
**PS-4** PORT SELECTOR (4 channels **RS-422)....$** 79.95
Converts an RS-232 port into 4 selectable W-422 ports.
CO-485 (RS-232 to **RS-422/RS-485 converter)......$ 44.95**

***EXPANDABLE...**expand your interface to control and monitor up to 512 relays, up to 576 digital inputs, up to 128 anal0g inputs or up to 128 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.
380 South Fifth Street, Suite 604
Columbus, Ohio 43215.5436

#105

ISO 9001 Certified

**Data Acquisition & Control Solutions**

Temperature Measurement & Control Solution Package – $895

**Key Features:**

- For J, K, S, T, B, R, E thermocouples
- 8 thermocouple channels, expandable to 32 channels
- 0.1° C resolution with 12-bit A/D
- Up to 250 samples per second
- 16 D/I and 16 D/O channels
- Powerful Windows SCADA software
- Icon based, no programming required

- ❑ General Purpose DAS Card ......$295
  8 A/D, 1 D/A, 16/16 DIO, wiring kit
- ❑ Multifunction DAS Card ...........$395
  16 A/D, 2 D/A, 32 DIO, counter
- ❑ High Performance DAS Card......$595
  100KHz, programmable gain
- ❑ 24 Ch Digital I/O Card ..............$100
- ❑ 24 Ch Digital Input w/Interrupt ...$180
- ❑ 32 Ch Digital I/O Card ..............$170
- ❑ 32 Ch Digital Input w/Interrupt ...$235
- ❑ 144 Ch Digital I/O Card ...........$295
- ❑ 8 Relay & 8 Digital Input Card ...$210
- ❑ High Speed Data Streaming Pkg. $595
  100KHz data streaming to disk
- ❑ PC Strip Chart Recorder Pkg. ...$695
  16 Channel recording at 15KHz
- ❑ Remote DA&C Starter Kit .........$495
  RS-485 based, up to 4000 feet

***FREE!***

*232-page Solution Guide for quality minded, budget conscious Engineers*

**1-800-800-6889**

*Industrial & Lab Automation with PCs*

**ADVANTECH®**

750 East Arques Ave., Sunnyvale, CA 94086
Tel: (408) 245-6678 FAX: (408) 245-8268

#106

## RING GENERATOR

The ring generator (Figure Id) is simply a small inverter that is used to step up the 24-volt DC unfiltered supply to 90 volts AC with a frequency that is controlled by the CPU. The frequency is normally 20 Hz to ring the telephone sets, however some telephone sets have message waiting lights that respond to a 40- or 80-Hz signal without ringing the bell, so the computer has the option of generating these alternate frequencies.

The ring generator circuit also has a loop current detector so if the computer is ringing the telephone set and the user picks up the receiver during a ring, the computer is notified to immediately stop the ring and connect the party to the caller. This "in-cycle" answer of a telephone is called ***ring trip.***

## COMMON CONTROL

The system's common control (Figure le) includes two dial tone generators and DTMF receivers that can be connected to any of the telephone sets through the switch matrix under the CPU's control whenever a telephone set demands service.

The dial tone generator chip is available from only one source: Teltone. Requiring just a 3.579-MHz crystal to operate, it generates all the industry-standard call progress tones including dial tone, ringback, busy, reorder, several intercept tones, a DTMF "#" (but no other DTMF signals), and others. Its downside is it's pricey, single sourced, and is not directly CPU bus compatible. If you care to, you may replace this device with a dedicated dial-tone circuit consisting of a few op-amps generating a 400-Hz sine wave and gate that into the point where the Teltone generator would have gone.

The dial tone generators are buffered through an op-amp and then coupled to the matrix through a 10-µF nonpolarized capacitor. The input to the DTMF receiver is connected to a rail in the switch matrix along with the output of the dial tone generator.

The SSI20C90 DTMF receiver also has call progress detection and a built-in DTMF generator. This device is very similar to the SSI202CP which has been used in several projects in the ***Computer Applications*** *Journal.* Requiring only a 3.579-MHz color burst crystal to operate and a single 5-volt supply, it is directly compatible with the CPU bus, but requires a separate I/O port to sense the "digit valid" and "call progress" bits. In some applications, these might instead go to the CPU interrupt structure.

The dial tone generator/DTMF receiver combinations are assigned on an as-needed basis. Once used, they are disconnected and may be used by another party. A conversation in progress between two telephone sets does not need a generator/receiver. Only two persons may be dialing at any given time, but there can be up to eight separate conversations happening at once.

## SWITCH MATRIX

The switch matrix is responsible for getting the analog signal from one place to another (Figure 1f). The device used in the Personal PBX project has 96 analog switches in an array of 12 x 8. Called the M093, it is available from a variety of manufacturers including Mitel and SGS-Thompson. The MO93 is powered from the 5-volt analog supply and will pass signals from any input to any output in either direction with only 75 ohms of resistance through any one crosspoint.

The switch matrix is controlled by the microprocessor and has all of the telephone sets connected to a different X rail in the device. The Y rails in the device are the speech paths and may be daisy-chained to multiple switch matrix chips to form a larger switch platform. To connect two telephones together, you close the X rail of the telephone sets on the same Y rail and the sets are then electrically connected in a conversation (see Figure 2).

One oddity of the MO93 device is that in addressing the X side of the crosspoints, it skips addresses 6 and 7. As a result, when referring to X0, you use address 0, but when referring to X6, you use address 8. This oddity is handled by the low-level library routines and is transparent when writing software for the switch.

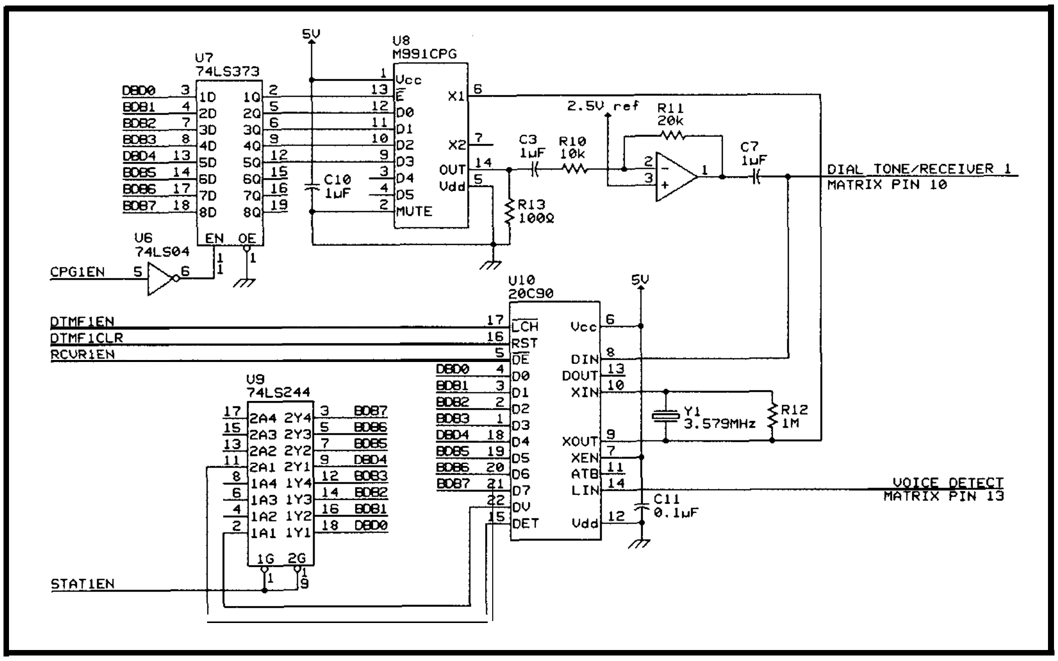


Figure 1e—*The* system's common control *includes two dial tone generators and DTMF receivers that can be connected to any of the telephone sets.*

In this project, I will use only one MO93 for now. Each of the rails X0–X7 connects to the SLICs 1-8, respectively. X8 and X9 are connected to dial tone generator/DTMF receiver combinations. X10 is connected to a speech playback device for recorded announcements, and Xl 1 is left available for experimentation.

## RECORDEDANNOUNCEMENT

An optional extra feature I haven't mentioned up to this point is a recorded announcement. In the full-blown commercial version of this switch, I use this feature to prompt the caller for a calling card number and keep them informed of the progress of the call. I selected the easy-to-use (though currently impossible to get) DAST device manufactured by Integrated Storage Devices.

The ISD2590 features 90 seconds of record/playback that, when accessed sequentially, requires only five control lines and two status lines. One nice feature of using the sequential message queuing mode is that very little CPU overhead is required to keep track of what this device is doing, so we can take our time in setting up the message to be played without unexpected results.

There is only one DAST in the system, but like all the other common peripherals here, it is assigned to the call only when it's being used and then is released to service other callers.

## THE SOFTWARE

The code that controls the Personal PBX is a large state machine that has been broken down into smaller C functions. The major functions include:

f_exhm( ): The hook switch monitor. Waits for a user to lift a phone, then sets up the phone to be serviced by other routines.

f_exdm( ): The dialing monitor. Everything having to do with dialing and call setup is handled here.

f_exsm( ): The extension signaling monitor. Ringing the telephone sets is this routine's job.

f_excm( ): The conversation monitor. During the call, this routine is in control, but its biggest job is cleanup when the call is finished.

The ma i n procedure contains the startup code. When the system is powered up or the processor is reset, the startup code immediately turns off the ringing generator coils, resets all relays to the off state, and clears the state machine variables to a known state.

When the dispatcher detects that it is time for a phone to get a share of the CPU time, it sets a global variable called e x t_c h e c k that equals the port being serviced and is also the index to the arrays that hold the state the telephone set is in.

Each phone in the system has memory variables in an array assigned to it. If there are eight phones in the system, then the array contains eight elements. The arrays ext_ process[x], ext_substate [ x ], and all others beginning with ex t_ are permanently assigned to the port. Normally the ex t_ variables are indexed with a global variable called d i spat ch-process. When port 3, for example, has a share of the CPU time, d i spa t ch__ process equals3.

There are other variables that are also arrays, but they are not permanently assigned to any one phone. Instead, they are assigned to a phone when that phone goes off-hook and they stay assigned during the

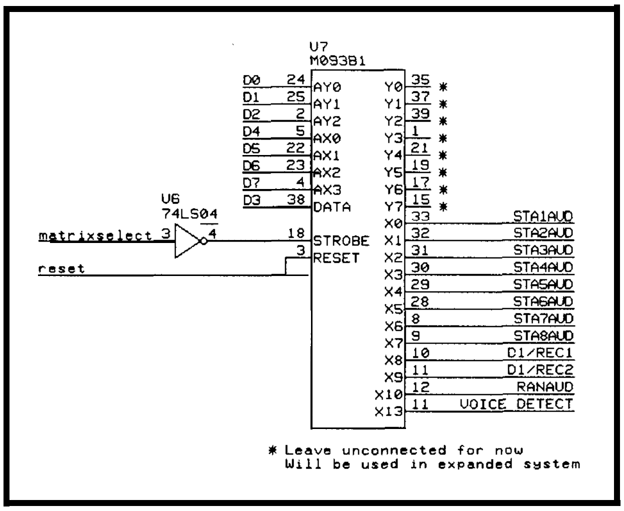


Figure 1f—*The switch matrix is responsible for getting the* analog *signal* from *one place to* another.

entire call. These are called *call control blocks* and are sets of arrays indicated by cc b_ before their name (e.g., cc b_ digits[x], ccb_acp [x ], etc.). Call control blocks are assigned to the phones by a port variable called ext_status[]. If CCB 1 is assigned to port 3,then ext_status[3] will equal 1.

Because each phone has a different index to the same set of variables, this state machine could be said to be servicing different *contexts* each time it is dispatched. Each context has very different concurrently executing functions. One phone might be dialing while another is ringing, but they are being serviced by the same software. You can expand the number of ports being serviced by making the array that holds the common variables larger.

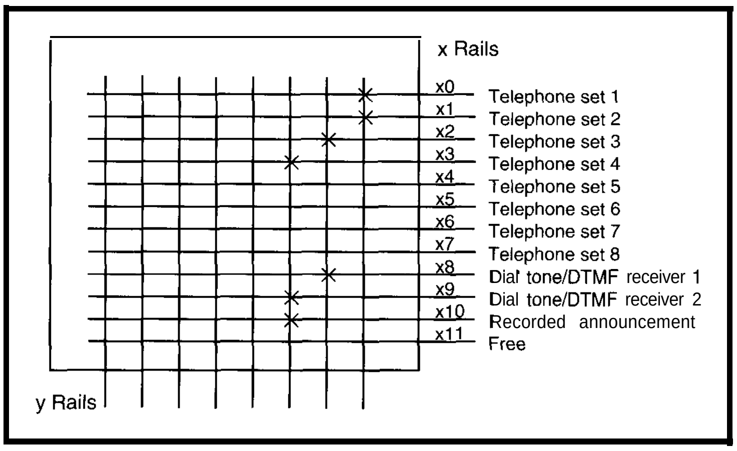


Figure **2**—*The* switch matrix *has all the telephone sets connected to the X rails while the Y rails are connected to speech paths.*

## SYSTEM OPERATION

The software attempts to emulate a central office, so it is transparent to a user dialing from a telephone connected to it. It might be helpful to follow a call through the state machine.

First, the source and destination telephone ports are idle. They are both executing state 0, which is looking at the current detector and waiting for it to detect off-hook.

When the person on port 0 picks up thephone, HookMonitor first locates a free call control block that will be used to store information about the call in progress. If HookMoni tor cannot find a free call control block, it simply does nothing and falls through to the end of the code block. The dispatcher then goes on to service all the other phones in the system. When it gets back around to port 0 again, it starts the same process of looking for a call control block. Ideally, it will always find a free call control block. In the worst case, it would find one when someone else hangs up and their block is freed.

Next, HookMoni tor clears the port's ext_route[] variable. This allows any process to see that the call has yet to go somewhere.

Thefinalsteps HookMoni tor takes are to locate a free dial tone sender, DTMF receiver, and speech path for the conversation to take place on. Once all of the above is done, the user hears dial tone from the earpiece of his telephone set. HookMoni tor now sets the extension state variable ext_process[] to33, which is the entrystateof DialingMonitor.

DialingMonitor decrementsthe extension timer, which gives the user a finite amount of time to dial the first digit. If nothing is dialed within this predetermined period of time, a reorder tone [fast busy) is sent to the port. When the reorder tone is sent, the same timer is loaded with a finite amount of time again. When it expires a second time, all of the resources assigned to the call are freed, including the call control block and speech path. There is now a phone off-hook, but no hardware assigned to it, so a *lockout condition,* or state 36, is assigned to it. This state does nothing more than wait for the phone to go on-hook, which will start the off-hook monitor all over again.

When DialingMonitor does detect a digit, it is stored in the cc b_ d igi t. s area and the routing tables are searched for a match. In my example, let's say the user dials a 4 first. After the 4 button is pressed and released, Dial i ngMoni tor will store the digit and then call se a r c h (), which will look in a single-digit table. In this case, s e a r c h () won't find a match for the 4 because there is nothing in the single-digit table to find. When the user then

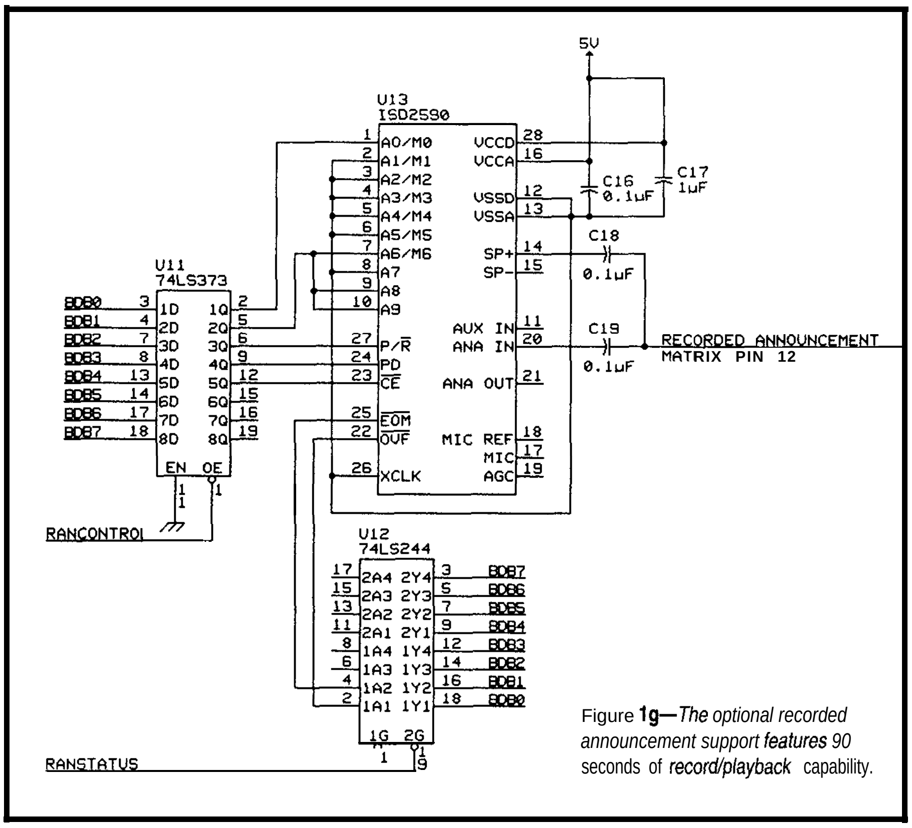


Figure **1g**—*The optional recorded announcement support features 90* seconds of *record/playback* capability.

dials a 3 as the second digit, search() will now look in a double-digit table, and will find a match for the 4 as the first digit and the 3 as the second digit. When this match is found, search() will update ext_route[] to be the action stored with the found digits. In our case, the action is to send the call to another port on the system.

When DialingMonitor sees that the time for a digit to be dialed has expired, it will look to see if a route has been chosen and, if so, whether it is a call to another port on this system.

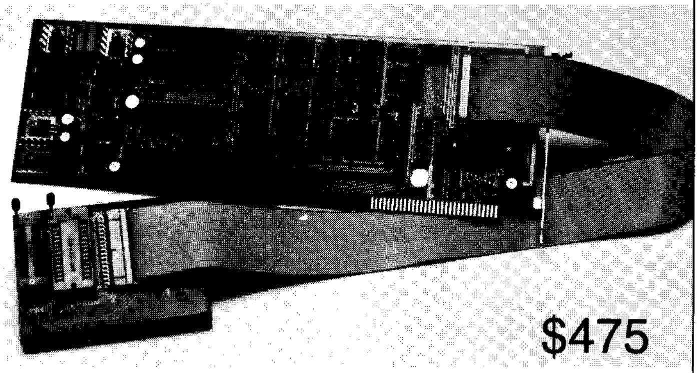
There is a reason I wait for the interdigit time to expire rather than just place the call as soon as I find a match. One of the original system design goals was to make this small system work transparently to its users, but still provide all of the features of a private branch exchange. In order to do this, I wanted to allow extension numbers to be able to conflict with local exchange numbers. For example, you might have a local telephone number like 423-5555 and at the same time support extension number 42. When the software sees that the dialed digits match 42, it selects the route to the other extension. If the user keeps dialing, the call would be sent to the local exchange via a trunk rather than to extension 42.

Since a decision has now been made where to send the call, DialingMonitor now looks to the extension variables for the selected destination extension. Since the process state of the destination extension is 0, we know the destination is idle and can change its state without interrupting a call. Changing its state to 66, which is the extension signaling monitor, we also change the station's ext_status[] variable to be the same as our own, which points to the call control block number we were assigned along with dial tone. Next, we change the state of our calling phone to be 37, which provides the caller with ringback tone.

Now the destination is ringing and the caller is listening to ringback. When the destination answers, the extension signaling monitor debounces the line by letting it sit for a moment without any ringing current on it. Then it connects the SLIC to the audio

## Universal Device Programmer

PAL
GAL
EPROM
EEPROM
FLASH
MICRO
87C51
PIC
93646
XC1 736
PSD 3xx
5ns PALs

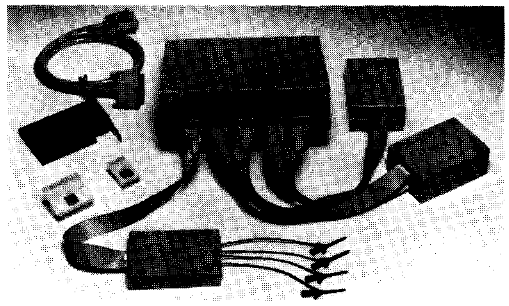


$475

Free software updates on BBS
Powerful menu driven software

## 400 MHz Logic Analyzer

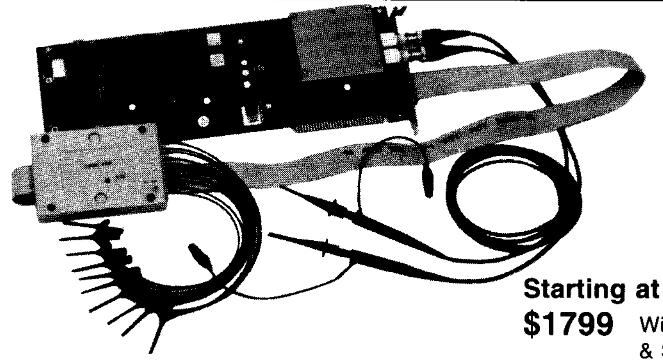
- up to 128 Channels
- up to 400 MHz
- up to 16K Samples/Channel
- Variable Threshold Levels
- 8 External Clocks
- 16 Level Triggering
- Pattern Generator Option



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

Price is Complete
Pods and Software
included

## 200 MSa/s DIGITAL OSCILLOSCOPE



Starting at
$1799 With Pods & Software

- 200 MSa/s sampling Rate
- 2 Analog Channels (2 ch. Digital Oscilloscope)
- 8 Digital Channels (8 ch. Logic Analyzer)
- 125 MHz Single Shot Bandwidth
- up to 128K Samples/Channel

Call (201) 808-8990

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

#107

path found in ext_status[] and changes the state to the conversation monitor, which is state 98.

Meanwhile, the calling station is still hearing ringback. While doing that, DialingMonitor is looking at the destination station's data, waiting for the state to be 98. When it sees that state, it knows the called party has answered. It then releases all the common controls, freeing them for others to use, and connects the calling extension to the same audio path. Finally, the calling station's state is also changed to 98.

At this point, we have a conversation and are both in state 98. The only thing the system has to do is wait for either caller to go on-hook and complete the call. When that happens, each station's state changes back to 0 to ready the ports for another call.

## THE SERIAL PORT

A serial port is provided for doing "OA&M" (Operations Administration and Maintenance). The code provided has modest requirements for administration. To change extension numbers, you must recompile the code.

Typing "N" from the serial port will load the RAM with the default extension numbers 40-45. It also loads an end-of-dialing digit so you can dial a "#" as the last digit and the call will be placed immediately rather than wait for the interdigit timer to expire.

## THE FUTURE

This system demonstrates basic telephone switching. The hardware could be used with most any computer controlling it. The software is very basic since, right now, it only supports internal extension calling.

We can expand this project to include external calling to the telephone company so our calls can reach the real world. To do so will require another interface between the switch matrix and the central office line pair.

The serial port is also awfully lonely. It could be attached to a PC to use the Personal PBX as a front end for voice response and remote control in a home. Stay tuned. ❑

*Richard Newman lives in Dallas, Texas where he is an independent engineer designing equipment for the telephone industry. He may be reached at (214) 522-8935.*

## SOURCE

A printed circuit board is available from the author. Contact him at (214) 522-8935 for more information.

## 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

401 Very Useful
402 Moderately Useful
403 Not Useful

**C COMPILER FOR PIC CONTROLLERS**

- Integrated software development environment including an editor with interactive error detection/correction.
- Access to all hardware features from C.
- Includes libraries for RS232 serial I/O and precision delays.
- Efficient function invocation mechanism allowing call trees deeper than the hardware stack.
- Special built-in features such as bit variables optimized to take advantage of unique hardware capabilities.
- Interrupt and A/D built-in functions for the C71.
- Easy to use high level constructs:

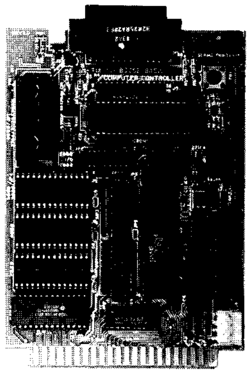
```
#include <PIC16C56.h>
#use  Delay(Clock=20000000)
#use  RS232(Baud=9600,Xmit=pin_1,RCV=pin_2)

main 0 {
   printf("Press any key to begin\n") ;
   getc();
   printf("1 khz signal activated\n");
   while (TRUE) {
      output_high(pin_8) ;
      delay_us(500) ;
      output_low(pin_8) ;
      delay_us(500);
    }
}
```

| | |
|---|---|
| C5x compiler | $99 (all 5x chips) |
| C-mid compiler | $99 ('64, '71, '84 chips) |
| Pre-paid shipping | $5 |
| COD shipping | $10 |

**CCS,** PO Box 11191, Milwaukee **WI 53211**
**414-781-2794 x30**

#108

**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 8-bit CMOS processor w/BASIC-52
- Three 16-bit counter/timers
- Six interrupts
- Much more!

**INPUT/OUTPUT**
*Console RS232 -autobaud detect
- Lineprinter RS-232
- Three 8-bit parallel ports
- EXPANDABLE1
  - 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** 89.00 Single Qty |
| **BCC52C** | Low-power CMOS version of the BCC52 | **$199.00** |
| **BCC52I** | -40°C to +85°C industrial temperature version | **$294.00** |
| **BCC 5 2CX** | Low-power CMOS. expanded BCC52 w/32K RAM | **$259.00** |

**CALL FOR OEM PRICING**

**MICROMINT, INC.** 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

FEATURE ARTICLE

**Hank Wallace**

# Using the Golay Error Detection and Correction Code

People often forget the work done long before the microprocessor when developing algorithms. The discoveries made by Marcel J.E. Golay in 1949 are well-suited for some of today's communications systems.

This is the scene: Smoke wafts slowly in the darkened room, as if with a mind of its own. Seedy characters sit at a dirty table under a naked bulb. At the focus, a Ouija board. Surrounding are piles of paper-data stacked upon data upon data, casting stark shadows. A sheet is snatched, examined, a scribble is placed on it, and then it's back to the pile. Occult bookies hard at work? No, error correction!

Well, that was my vision of it, until I was forced to bone-up for a recent communications project. I found out that error correction is not really like what you see in the movies: smoky mirrors, crystal balls, and evil mathematicians. If your vision of error correction is something to be avoided, I would like to put it in a better light and help you change that image.

Most engineers are familiar with basic methods to verify the integrity of data transmitted over noisy communications channels. Simple checksums and cyclic redundancy checks (CRCs) are the most popular. Presented here is a related coding technique by which errors cannot only be detected, but removed from received data without retransmissions. This code bears the name of its discoverer: Marcel J. E. Golay. The earliest reference to his work I found is a note he published in a 1949 technical journal [1]. Then, there was much activity in information coding techniques spurred by Claude Shannon's 1948 landmark work *The* Mathemati*cal Theory Of Communication.* The codes that Golay discovered can enhance the reliability of communication on a noisy data link.

Before we proceed, a note of caution. The theory of error detection and correction is deep (but not necessarily dark), as you can verify from any of the references cited. This article is intended only to give a brief overview of one of the many codes available. We will not dwell on messy mathematical details or terse comparisons of error correction techniques. If your system design calls for error correction methods, the C program fragments here may help you evaluate the Golay code. Just keep in mind that this is not a theoretically thorough treatment of this almost bottomless, fascinating subject.

We should get a couple of simple definitions out of the way to make what follows easier to digest.

*Weight:* The count of bits which are ones in a binary word. For example, the weight of the byte 01011011 is five since it contains five ones.

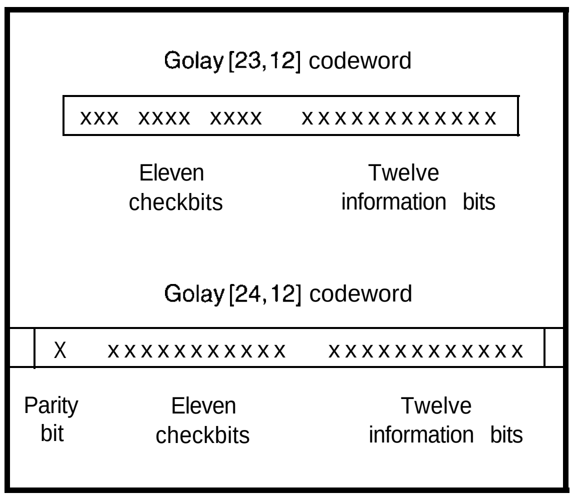


Figure 1—*The Golay codeword structure is* derived *from modulo-2* division, similar to the CRC.

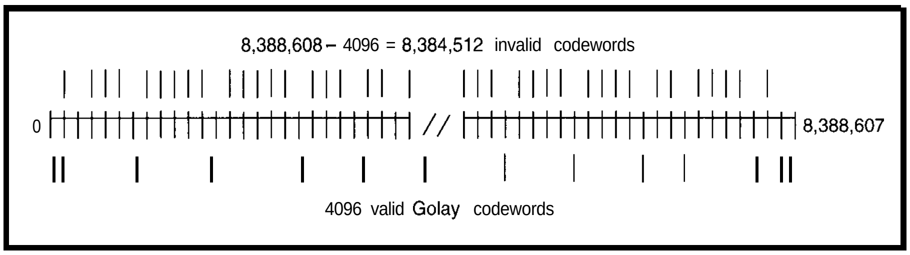

8,388,608 – 4096 = 8,384,512 invalid codewords
0
//
8,388,607
4096 valid Golay codewords

Figure 2—*Golay [23,12] codeword distribution on the number line spaced with* respect to the Hamming distance

***Hamming Distance:*** The number of bits which differ between two binary numbers. If A and B are binary numbers, the distance is weight(A XOR B). For example, the Hamming distance between bytes 4Ah and 68h is weight(4Ah XOR 6Fh) = weight(22h) = 2 bits.

***Forward Error Correction:*** A technique which can correct errors at the receiver without relying on retransmissions of data.

## NUTS AND BOLTS

Let's jump right in and examine the structure of the binary Golay code. A ***codeword*** is formed by taking **12** information bits and appending 11 check bits which are derived from a modulo-2 division, as with the CRC. See Figure 1. We will examine the modulo-2 division process later. The common notation for this structure is Golay[23,12], indicating that the code has 23 total bits, 12 information bits, and 23–12=11 check bits. Since each codeword is 23 bits long, there are $2^{23}$, or 8,388,608, possible binary values. However, since each of the 12-bit information fields has only one corresponding set of 11 check bits, there are only $2^{12}$, or 4096, valid Golay codewords. You can think of the codewords as being distributed along a number line. See Figure 2.

The Golay codewords are not really spread evenly as you count along the number line, say every few ticks. Rather, they are spaced with regard to the Hamming distance between them. It turns out that each Golay codeword has seven or more bits differing from every other. Mathematicians say that the code has a minimum distance, ***d,*** of seven. They have determined analytically that the Golay code can detect and correct a maximum of (d-1)/2=3 bit errors, in any pattern.

To augment the power of the Golay code, an overall parity bit is usually added, resulting in a clean 3-byte codeword called the ***extended*** *Golay* **code,** noted as Golay[24,12]. With this parity bit, all odd numbers of bit errors can be detected in each codeword, as well as all 4-bit errors. We'll get to the nuts and bolts of encoding and decoding codewords presently.

Before you trash all your old CRC-based designs, be advised that there is a tradeoff associated with the Golay code's error correction ability. If the code is used merely to detect errors, it can find a maximum six of them in any codeword. However, if correction is performed, only three bits are correctable. Thus, we trade identifiable errors for correctability. As the code is used in an application, situations may demand changing the correction/detection methods to suit. Keep this tradeoff in mind as you examine the performance of the Golay code and the requirements of your application.

Let's look at the error-trapping ability of the code. If error correction is not attempted, the following are the error-detection-only properties per 24-bit extended Golay codeword:

100% of 1- to 6-bit errors detected, any pattern
100% of odd bit errors detected, any pattern
99.988% of other errors detected

Using the error correction facilities of the code, these are the data reliability rates:

100% of 1- to 3-bit errors corrected, any pattern
100% of 4-bit errors detected, any pattern
100% of odd numbers of bit errors detected, any pattern
0.24% of other errors corrected (1/4096)

The parity bit of the extended Golay[24,12] code augments its error-corrective properties, allowing all combinations of 4-bit errors to be detected, but not corrected.

Error correction is obviously no panacea and does carry a penalty. Let me explain why. When a codeword is being corrected, the correction algorithm looks for the closest matching codeword. For example, say codeword E86555h is transmitted (data=555h, checkbits=E86h), but four bit errors occur, corrupting the codeword to E86476h. When we feed this codeword into the correction function, it returns 68E4E6h as the closest matching codeword, not E86555h. The receiver can use the parity bit to detect this error, but it cannot detect a higher even number of errors. This illustrates the fact that every correction algorithm has a bit-error-rate (BER) limit, beyond which it cannot compensate. Fortunately, you may enable the correction facilities of the given Golay C routines according to your needs.

Figure 3 illustrates what is happening during codeword correction. We see three valid Golay codewords: A, B, and C. The X-axis represents

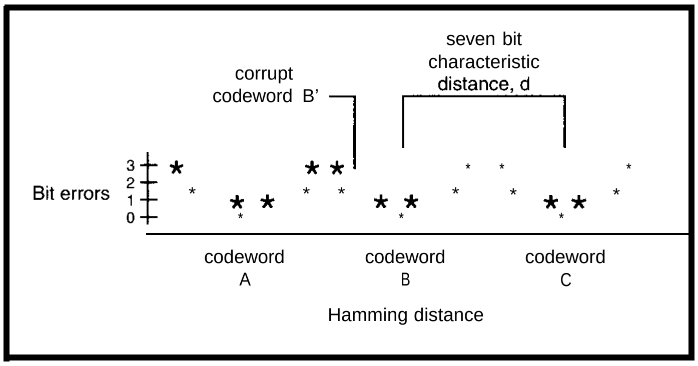
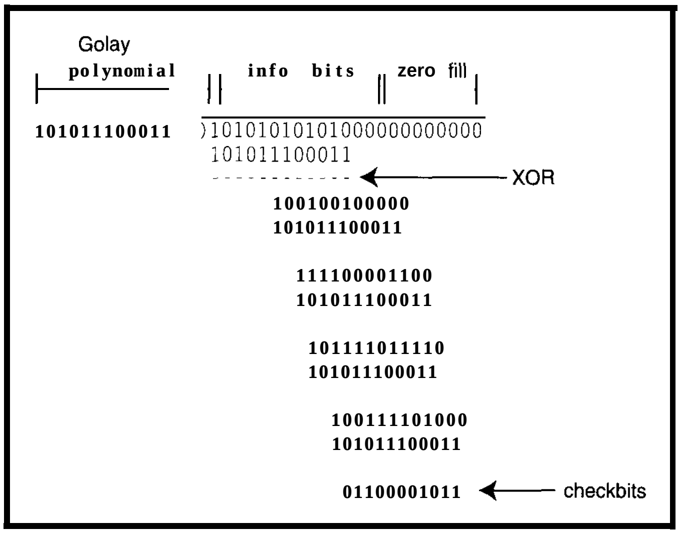


Figure 3—*Simplified illustration of Golay [23,12] error correction for three valid codewords.*

Hamming distance between these codewords. The Y-axis represents the number of bit errors incurred as you move away from a valid codeword. Picture the correction algorithm as taking the input codeword, say at point B', and sliding down the slope to the nearest correct codeword. If B' is a corrupted version of codeword B, we're in luck. If B' is a very corrupted version of A, the correction algorithm lies to us and returns B as the corrected codeword. You can see that the minimum distance, *d,* of the code controls the amount of corruption that can be tolerated. This is a simplified explanation, of course. In reality, this diagram is multidimensional and each Golay codeword has many error-laden companions to keep it company in the abstract boredom of n-space.



```
101011100011 )10101010101000000000000
              101011100011
              ------------
                  100100100000
                  101011100011

                   111100001100
                   101011100011

                    101111011110
                    101011100011

                     100111101000
                     101011100011

                      01100001011
```

Figure 4-Calculation *of Golay checkbits using modulo-2 division.*

Now for some mathematical curiosities. Among error correction codes, there is a class known as *perfect codes.* Briefly, perfect codes are defined as those where each of the invalid codewords shown on the number line in Figure 2, when pumped through the correction process, will be transformed into a valid codeword. There are no ***orphan uncorrectable information vectors.*** Included as perfect codes are the ***Hamming codes,*** a one-bit correction scheme, and the ***binary*** and ***ternary** Golay* codes. The Golay codes are the only nontrivial multiple-error-correcting perfect codes that exist [3]. The perfect quality of the Golay code makes it an object of beautyintheeyesof mathematicians, especially when this property is expressed in terms of the optimum packing of spheres into a region of space [4]. This perfection also makes the Golay code useful in the hands of communications engineers.

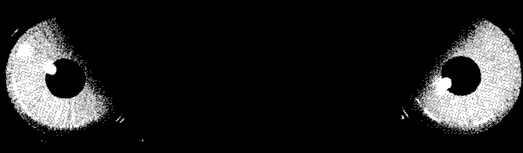
The Golay codeword has some very interesting properties:

*Cyclic Invariance. If you take a 23-bit Golay codeword and cyclically shift it by any number of bits, the result is also a valid Golay codeword.

•Inversion. If you take a 23-bit Golay codeword and invert it, the result is also a valid Golay codeword.

*"Is the*

**TERROR**



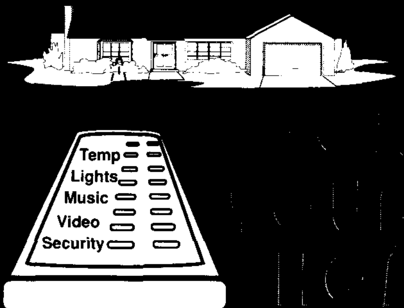
***of BIG S-L-O-W software stalking your job security?"***

**Take some of the bytes out of your bloated code!**
**Call Franklin Software, the first and last word in quality for your 8051 and 80C166 software projects!**

**FRANKLIN SOFTWARE, INC**

***Call for your free Release VI Evaluation Kit!***
**tel: (408) 296-8051 fax: (408) 296-8061**

#110



[illegible] HOME

[illegible] SYSTEMS, INC

[illegible] Source of Home Control Products

[illegible] Selection of X10 Compatible Products In the World

[illegible] Guaranteed Lowest Prices

[illegible] NOW for FREE CATALOG

[illegible] SMART-HOME *(800-762-7846)* 24hrs

Over 300 hard-to-find home control products you can install yourself. Security, music, video, heating/AC, lighting, surveillance and more. No rewiring. New or existing homes. Affordable - systems start at under $20. Catalog has detailed explanations and amazing project ideas.

VISA MasterCard AMERICAN EXPRESS

151 Kalmus Dr., Suite M6, Costa Mesa, CA 92626
Questions (714) 708-0610 Fax (714) 708-0614

#111

**Listing 1**--The *Golay [23,12]* codeword *encoder returns a long integer with fhe format [checkbits(11), data(12)].*

```
#define POLY 0xAE3 /* or use the other polynomial, 0xC75 */

unsigned long golay(unsigned long cw)

  int i;
  unsigned long c;
  cw &= 0xfffl;
  c = cw;                     /* save original codeword */
  for (i=1; i<=12; i++){      /* examine each data bit */
    if (cw & 1)               /* test data bit */
      cw ^= POLY;             /* XOR polynomial */
    cw >>= 1;                 /* shift intermediate result */

  return((cw<<12) | c);       /* assemble codeword */
```

*Minimum Hamming Distance. The distance between any two Golay [23,12] codewords is always seven or more bits. The distance between any two Golay [24,12] codewords is always eight or more bits.

*Error Correction. The correction algorithm can detect and correct up to three bit errors per codeword.

## WHAT'S THE USE?

The Golay code is obviously not able to encode a large amount of data in one codeword. Twelve bits is the maximum allowed. So what is its use? Well, one advantage to error correction is the elimination of communications retries which can bog down a noisy channel. In some cases, the overhead associated with a resend request is much greater than the length of a data packet. For example, to send a data packet on a half-duplex or simplex radio system, the microprocessor must turn on the transmitter and wait for it to come up to full power, typically 100 ms. The originating station must incur the same pretransmit delay when resending its data. Even if all goes well, there is 200 ms of wasted time involved in a resend request. At 1200 bits per second, 200 ms represents 240 bits of data, a reasonable message length in some systems. And if the radio channel is busy with other traffic, the delays can be longer. The

**Listing 2**—*The parity bit generator checks the overall* parity *of* codeword cw. *If parity* is even, a *0* is *returned; otherwise, a 1 is returned.*

```
int parity(unsigned long cw)


  unsigned char p;

  /* XOR the bytes of the codeword */
  p = *(unsigned char*)&cw;
  p ^= *((unsigned char*)&cw+1);
  p ^= *((unsigned char*)&cw+2);

  /* XOR the halves of the intermediate result */
  p = p ^ (p>>4);
  p = p ^ (p>>2);
  p = p ^ (p>>1);

  /* return the parity result */
  return(p & 1);
```

What is the
**80C552**

It's a high integration, 8051 with:

| | |
|---|---|
| **8 ch. 10 bit A/D** | **2 PWM outputs** |
| **Cap/cmp registers** | **16 I/O lines** |
| **RS-232 port** | **Watchdog** |

**W'e've** made the 552SBC by adding:

**2-RS-232/485 multi-drop ports**

| | |
|---|---|
| **24 more I/O** | **Real-time Clock** |
| **EEPROM** | **3-RAM & 1-ROM** |
| **Battery Backup** | **Power Regulation** |
| **Power Fail Int.** | **Expansion Bus** |

Start with the Development Board - all the peripherals, power supply, manual and a debug monitor for only $349. Download your code and debug it right on this SBC. Then use the **$149** and up OEM boards for production. or have us make a custom board for you. Call now for a brochure!

**188SBC**

**Use Turbo or MS 'C'**

**Intel 80C188XL**
**Two 1 meg Flash/ ROM sockets**
**Four battery backed, 1 meg RAM**
**16 channel, 12 or 16 bit A/D**
**8 channel, 12 bit D/A**
**2 RS-232/485 serial, 1 parallel**
**24 bits of opto rack compatible I/O**
**20 bits of digital I/0**
**Real-time clock**
**Interrupt and DMA controller**
**8 bit, PC/104 expansion ISA bus**

Power on the quiet, 4 layer board is provided by a switcher with watchdog and power fail interrupt circuitry.

**It's got "E I E I/O"**

The **188SBC** is also available with Extended Interface Emulation of I/O a Xilinx **Field Programmable Gate Array** and a breadboard area. You can now define and design nearly any extra Interface you need. 188SBC prices start at $299. **Call right now** for a brochure!

$149 8032 ICE
Still *A vailable!*
8031 SBC as low as $49

**Call for your custom product needs. Quick Response.**

VISA MasterCard

HTE
**HiTech Equipment Corp.**
**9400 Activity Road**
**San Diego, CA 92126**
**[Fax: (619) 530-1458]**
Since 1983
— (619) 566-1 892 —
70662.1241 **@compuserve.com**

#112

Golay code can reduce the number of retransmission events by allowing the receiving end to correct some errors in the received data, decreasing the probability that the channel will get overloaded.

In other situations, there may be no resend request possible (e.g., with a one-way infrared or ultrasonic data link). The Golay code allows such systems to increase the probability of one-way, error-free reception.

Some communications channels are more prone than others to burst errors, where many consecutive data bits are corrupted. The Golay code alone is not able to correct bursts of errors over three bits long in a single codeword. However, when augmented with a technique called *bit interleaving,* the Golay code's small, modular format allows the correction of large burst errors, as we will see. The unmodified Golay code really shines in applications prone to random bit errors though, and it tolerates a disgusting (3 bit errors)/(24 bits per codeword) = 12.5% bit error rate without data retransmissions.

Of course, a disadvantage of the extended Golay code is that you must transmit as many check bits as data bits. Maybe you are asking, why not just send the data twice, without check bits? The answer is that no error correction is possible in such a system. How would you know which of two different messages, if not both, were corrupt? The Golay code allows error correction in exchange for the data-doubling price.

## IMPLEMENTATION-ENCODING

The Golay code is encoded just like the CRC, using modulo-2 division. However, there are only 12 information bits per codeword, so you must break your data down into 12-bit chunks and encode each as one codeword. The characteristic polynomials for the Golay code are:

```
   Golay
polynomial                         checkbits
                                |             |
101011100011 )10101010101001100001011
              101011100011

                   100100101100
                   101011100011

                     111100111100
                     101011100011

                       101110111111
                       101011100011

                         101011100011
                         101011100011

                          00000000000  f---syndrome= 0
```

Figure 5-*The syndrome is the remainder left after the codeword is divided by the generating* **polynomial,** *modulo-2. If the* **Golay codeword is valid, the syndrome is zero**

1. $X^{11} + X^9 + X^7 + X^6 + X^5 + X + 1$, coefficients AE3h.
2. $X^{11} + X^{10} + X^6 + X^5 + X^4 + X^2 + 1$, coefficients C75h.

Yes, there are two. You only need to use one of them because they both have the same properties as mentioned above, though they generate differing checkbits. Pull out your favorite buffalo nickel and choose a polynomial! We will use the coefficients of the X terms of the first polynomial, AE3h. (Note that AE3h is the bit-reversed version of C75h. There is a greater power at work here.)

An example worked out by hand illustrates the modulo-2 division process. Remember that in modulo-2 division, we XOR instead of subtract. The data we wish to encode is 555h. To generate the 11 check bits, append 11 zeros onto the bit-reversed data (least-significant bit first) and perform the division in Figure 4.

We only care about the bit-reversed remainder from the division, 11010000110 = 686h, so do not even write down the quotient. Putting the codeword together for transmission, we get 686555h. A parity bit could be added to the codeword to make an extended Golay code. Of course, you can mish-mash the bits around in any order for transmission, just as long as they are reassembled correctly before doing the decoding.

The encoding algorithm is shown in Listing 1. Long integers are used to conveniently store one codeword. The routine `parity()` adds the parity bit to complete the extended codeword. It is shown in Listing 2. You can see reference 3 for matrix methods which encode a Golay[24,12] codeword directly without an explicit parity routine.

## IMPLEMENTATION—DETECTING ERRORS

Detection of errors in a codeword is easy. First, we use the overall parity bit to check for odd numbers of errors, possibly failing the codeword on that basis. If parity is correct, we compute the 23-bit syndrome of the codeword and check for zero. The syndrome is the remainder left after the codeword

Listing 3-A *Golay [23,12] codeword syndrome generator. A table of intermediate results could also be used to speed the process.*

```
unsigned long syndrome(unsigned long cw)

{
  int i;
  cw &= 0x7fffffl;
  for (i=1; i<=12; i++) { /* examine each data bit */
    if (cw & 1)           /* test data bit */
      cw ^= POLY;         /* XOR polynomial */
    cw >>= 1;             /* shift intermediate result */

  return(cw<<12);         /* value pairs with upper bits of cw */
```

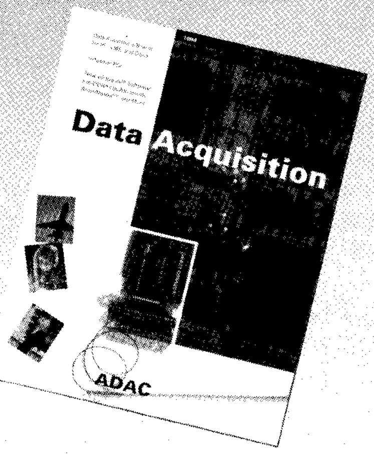
Listing 4—*Golay [23,12] codeword correction routines.*

```
int weight(unsigned long cw)

/* Calculate the weight of 23-bit codeword cw. */
{
  int  bits,k;

  /* nibble weight table */
  const char wgt[16] = {0,1,1,2,1,2,2,3,1,2,2,3,2,3,3,4}

  bits = 0; /* bit counter */
  k = 0:

  /* do all bits,  six nibbles max */
  while  ((k<6) && (cw)){
    bits = bits+wgt[cw & 0xf];
    cw >>= 4;
    k++;
  }
  return(bits);
}


unsigned long rotate_left(unsigned long cw, int n)

/* Rotate 23-bit codeword cw left by n bits. */
{
  int i;

  if (n != 0){
    for  (i=1; i<=n; i++){
      if ((cw & 0x4000001) != 0)
        cw  =  (cw << 1)|1;
      else
        cw <<= 1;
    }
  }
  return(cw & 0x7fffff1);
}


unsigned long rotate_right(unsigned long cw, int n)

/* Rotate 23-bit codeword cw right by n bits. */
{
  int i;

  if (n != 0)
    {
      for  (i=1; i<=n; i++)
        {
          if ((cw & 1) != 0)
            cw=(cw >> 1)|  0x4000001;
          else
            cw>>=1;
        }
    }
  return(cw & 0x7fffff1);
}


unsigned long correct(unsigned long cw, int *errs)

/* Correct Golay [23,12] codeword cw, returning the corrected
codeword. This function will produce the corrected codeword for
three or fewer errors. It will produce some other valid Golay
codeword for four or more errors, possibly not the intended one.
*errs is set to the number of bit errors corrected. */
```

*(continued)*

**NEW Data Acquisition Catalog**

***Covers expanded line.***



**FREE!**

1994 120 page catalog for PC, VME, and Qbus data acquisition. Plus informative application notes regarding anti-alias filtering, signal conditioning, and more.

**NEW Software:**
**LabVIEW ®, LabWindows®, Snap-Master™, and more**

**NEW Low Cost I/O Boards**

**NEW Industrial PCs**

**NEW Isolated Analog and Digital Industrial I/O**

**New from the inventors of plug-in data acquisition.**

**Call, fax, or mail for your free copy today.**

*ADAC*

American Data Acquisition Corporation
70 Tower Office Park, Woburn, MA 01801
Phone: (800) 648-6589 Fax: (617) 938-6553

#113

is divided by the generating polynomial, modulo-2. If the codeword is a valid Golay codeword, the syndrome is zero, otherwise it will be nonzero. Figure 5 shows a sample calculation using the previously constructed codeword, 686555h, bit reversed for the division process.

The routine in Listing 3 does the same sort of calculation. You could use a table of intermediate results to speed the process. These routines are all you need to implement a Golay error detection scheme. As above, you can detect up to six bit errors per codeword and, with the parity bit, all odd numbers of errors.

## IMPLEMENTATION—CORRECTING ERRORS

Cue the smoke and mirrors; error correction is not so trivial. It relies on the cyclic invariant properties of Golay codewords in a scheme called *systematic search decoding* [2]. There are other methods of Golay error correction listed in the references, though I have found this one easy to implement in software. This is a "nearest neighbor decoder," because it determines which Golay codeword is closest in terms of Hamming distance.

Here is a sketch of the systematic search algorithm.

1. Compute the syndrome of the codeword. If zero, no errors exist so go to step 5. If a trial bit has been toggled, go to step 2a, else go to 2.

2. If the syndrome has a weight of three or less, the syndrome matches the error pattern bit-for-bit and can be used to XOR the errors out of the codeword; if so, remove the errors and go to step 5, otherwise proceed to step 3.

2a. If the syndrome has a weight of one or two, the syndrome matches the error pattern bit-for-bit and can be used to XOR the errors out of the codeword; if so, remove the errors and go to step 5, else proceed.

3. Toggle a trial bit in the codeword in an effort to eliminate one bit error. Restore any previously toggled trial bit. If all 23 bits have been toggled and tried once, go to step 4; else go to step 2a.

4. Rotate the codeword cyclically left by one bit. Go to step 1.

5. Rotate the codeword right to its original position, if needed.

The idea is to fiddle with the codeword until the syndrome has a weight of three or less, in which case we can XOR the syndrome with the codeword to negate the errors. However, if a trial bit has been toggled, we might have introduced an error (making a total of four), so the threshold for XORing the errors away in step 2a must be reduced by one for safety, to two or less.

We are relying on the perfect nature of the Golay code for a terminating condition of the search algorithm. Normally, we would test in step 4 to see if all 23 codeword shifts had already been tried, but since the Golay code is perfect, we know that each 23-bit value maps to exactly one correct Golay codeword and the algorithm will terminate. However, since programmers are not perfect, I test to guard against infinite loops.

Listing 4—*continued*

```
  unsigned char w;  /* current syndrome limit weight, 2 or 3 */
  unsigned long mask: /* mask for bit flipping */
  int i,j;          /* index */
  unsigned long s,  /* calculated syndrome */
   cwsaver:         /* saves initial value of cw */

  cwsaver = cw;     /* save */
  *errs = 0;
  w = 3;            /* initial  syndrome weight threshold */
  '  = -1:          /* -1 = no  trial bit flipping on first pass */
  mask = 1:

  wh le (j < 23) {           /* flip each trial bit */
     f (j != -1){            /* toggle a trial bi t */
      if  (j > 0){           /* restore last tria1 bit */
        cw= cwsaver ^ mask:
        mask += mask:        /* point to next bit */

      cw = cwsaver ^ mask; /* flip next trial bit */
      w = 2;          /* lower the threshold whi le bit diddling */


    s = syndrome(cw);                 /* look for er^ors */

    if  (s){                          /* errors exist */
      for  (i=0; i<23; i++){
        /* check syndrome of each cyclic shift */
        if ( *errs=weight(s)) <=  w){
          /* syndrome matches error pattern */
          cw = cw ^ s;                /* remove errors */
          cw = rotate_right(cw,i);    /* unrotate data */
          return(s=cw);
        }
        else {
          cw = rotate-left cw,1);     /*  next pattern */
          s = syndrome(cw)            /* calc new syndrome */


      j++;              /* toggle next trial bit */

    else
      return(cw);       /* return corrected codeword */

  return(cwsaver);      /* return original if no corrections */
}/* correct */
```

Listing 5—*Golay [24,12] codeword decoder. This function decodes codeword *cw in one of two modes. If correct_mode is nonzero, error correction is attempted, with *errs set to the number of bits corrected, and returning 0 if no errors exist or 1 if parity errors exist. If correct_mode is zero, error detection is performed on *cw, returning 0 if no errors exist, 1 if an overall parity error exists, and 2 if a codeword error exists.*

```
int decode(int correct_mode, int *errs, unsigned long *cw)
{
  unsigned long parity-bit;

  if (correct_mode){                /* correct errors */
     parity-bit = *cw & 0x8000001; /* save parity bit */
     *cw &= -0x8000001; /* remove parity bit for correction */

     *cw=correct(*cw, errs);       /* correct up to three bits */
     *cw|=parity_bit;              /* restore parity bit */

     /* check for 4 bit errors */
     if (parity(*cw))              /* odd parity is an error */
       return(1);
     return(0); /* no errors */
  }

  else {/* detect errors only */
    *errs=0;
    if (parity(*cw)){/* odd parity is an error */
      *errs=1;
      return(1);
    }
    if (syndrome(*cw)){
      *errs=1;
      return(2);
    }
    else
      return(0); /* no errors */
  }

}/* decode */
```

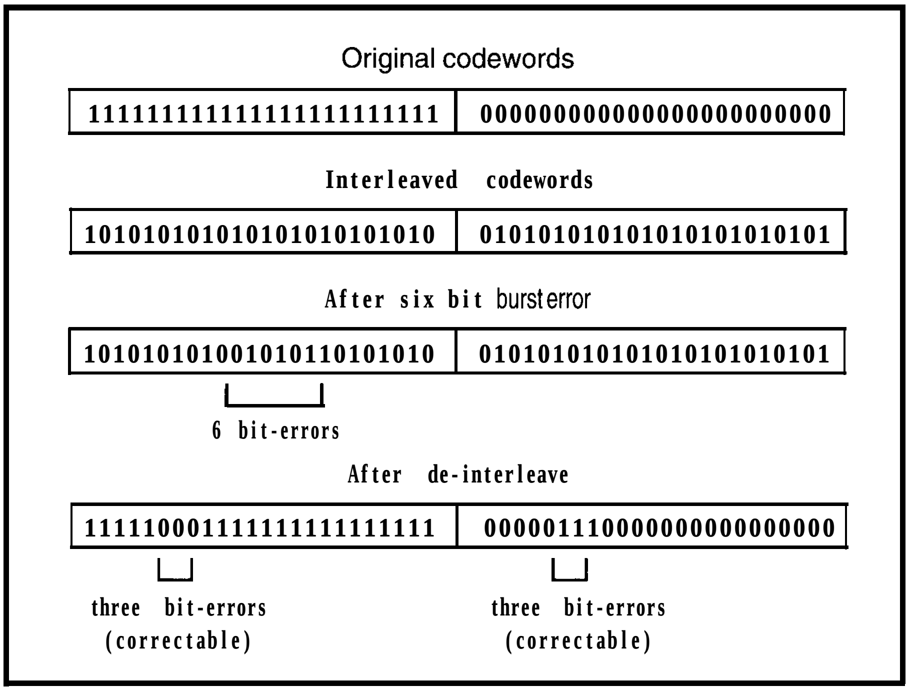


Figure 6—*Bit interleaving can be used in the presence of burst errors to extend the effectiveness of the Golay method.*

The rotations and trial-bit bookkeeping make for a nontrivial-looking routine for Golay error correction, correct(), shown in Listing 4 with its support routines. These routines are written for simplicity. You can tighten them up for optimum performance on your favorite scream machine.

The correct.()function returns the nearest matching Golay codeword. Note that this may not be the codeword you expect if there are more than three bit errors because the received, corrupted codeword may be closer, in terms of Hamming distance, to a different Golay codeword.

Listing 5 shows decode( ),a function that handles either detecting or correcting errors in Golay [24,12] codewords, returning the corrected codeword and error detection status upon exit.

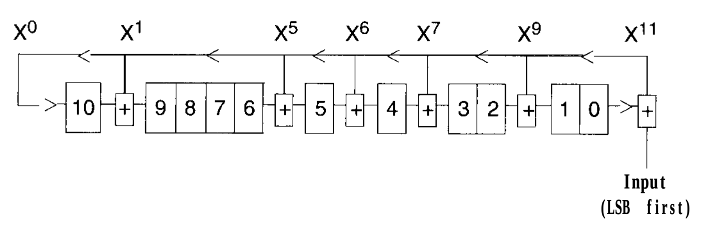
The parity bit in the extended Golay code is used as a final check on the integrity of the corrected codeword. If the received codeword has the right parity, the codeword is accepted, otherwise it is rejected. The overall parity bit is used to trap the occurrence of odd numbers of errors and four-bit-error-laden codewords. That the overall parity bit traps four-bit errors is explained by the fact that the correction algorithm introduces three additional bit changes to get to the nearest codeword, which is seven bits distant from the original, noncorrupt codeword. So, the four-bit errors, plus three additional changed bits, make seven bit changes total.

Note that toggling any bit(s) in a codeword an odd number of times will change the codeword's parity, allowing the overall parity check to trap four-bit errors, but only after correction has been attempted. If one of the four errors is in the parity bit, the 23-bit codeword will be corrected properly, but must be trashed because the total parity is wrong and the receiver will not know exactly which bits were corrupted. For the interested few, this will occur in $[C(24,4)-C(23,4)]/C(24,4) = (10626-8855)/10626 = 1/6$ of cases of four-bit errors, where $C(n,r)$ is the number of combinations of $n$ things taken $r$ at a time.

## BACK TO THE REALITY

In practice, I have used the Golay code on radio channels and telephone lines and have found it to be robust. Practically, one cannot rely on the error correction facilities alone if there is a two-way communication link. Rather, I use the Golay code to reduce the incidence of message resends. Messages are always checked for framing errors and other problems which are specific to the information transmitted and the message context.

A frailty of the Golay code is its inability to detect large bursts of bit errors. However, a technique called *bit interleaving* can remedy the situation. Assume we have a block of 48 bits to transmit-two codewords. An error of four consecutive bits will corrupt a Golay data packet and elicit a retransmission. However, if we transmit the 48 bits not in two sequential Golay codewords, but in a scrambled order, burst errors will be distributed over both codewords when the bits are unscrambled. See Figure 6.



```
                Shift register        Shift register
                Bit10 ┐               ┌ Bit 0
                                         ┌ input data,
                                           LSB first
initialregistervalue —>  00000000000
                   ┌  1  11000111010   1
                   │  2  01100011101   0
                   │  3  00110001110   1
                   │  4  00011000111   0
                   │  5  00001100011   1
value after shift— │  6  11000001011   0
                   │  7  01100000101   1
                   │  8  11110111000   0
                   │  9  10111100110   1
                   │ 10  01011110011   0
                   │ 11  00101111001   1
                   └ 12  11010000110   0
```

**Figure 7**—*Golay checkbit generator and example for polynomial $x^{11}+x^9+x^7+x^6+x^5+x+1$. Modulo-2 division example is shown in figure 5.*

The two resulting codewords in Figure 6 are correctable, whereas a six-bit burst error in the original data would not have been. In general, the more data you interleave, the longer the burst of errors the message can withstand. The correctable burst size, in bits, is three times the number of codewords interleaved. The penalty is the reception delay incurred between the first and last bit of any one codeword. Using this method, it is theoretically possible to extend the 3/24=12.5% burst error correction capability of the Golay[23,12] code to arbitrarily long blocks of data. Imagine sending a 100-codeword, 2400-bit message and correcting a burst error of 300 consecutive bits! Unfortunately, interleaving data in this fashion is not fun or memory efficient, but that's life.

You may have noticed that the codewords used in the above examples are OOOOOOh and FFFFFFh. These are valid Golay codewords. This alerts us to a possible communications failure mode: the stuck bit. If the hardware were to fail in midmessage, it could emit a default 1 or 0, depending on its configuration and the data format. This could be disastrous if undetected because the dead circuit would be interpreted as sending consecutive, valid Golay codewords. But happily, we can modify the Golay checkbits slightly, as is often done when using CRCs, by inverting one or more of them before transmission, and reinverting the same bits before decoding at the receiver. This way, OOOOOOh and FFFFFFh are not valid Golay codewords, and the chances of a stuck data line mimicking good data are greatly reduced. In fact, it is always good practice to structure your communications protocol so that a continuous 0 or 1 block of data cannot be misconstrued as valid information.

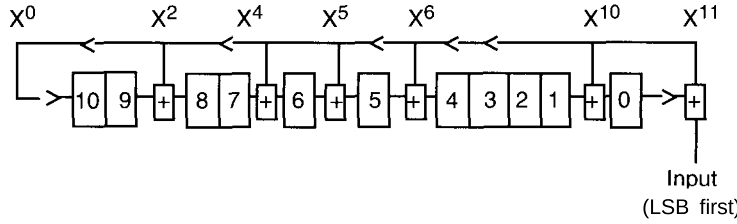
By the way, do not ignore the information that spins off of the correction process: The number of errors corrected in each codeword. This is valuable information about the BER on the communications channel, useful for judging its quality. Say you have a data system running over the telephone network in a remote area and these lines are not the best quality, degrading some with the weather. This is not a contrived example. Some Mom and Pop telcos run with twine and tin cans. It would be nice to optimize the data rate based on the current condition of the lines. Counting corrected bit errors is one way to do this. As the error rate (corrected bits / total bits) passes a critical value, the data rate can be cut, with the cooperation of transmitter and receiver, of course. When the sun comes out, the data rate can be increased as the BER goes down to acceptable levels. (Note that this technique has a BER measurement ceiling of 12.5% because every corrupt codeword will have at most three correctable errors.)

I am sure some readers are hardware types, wondering if the Golay code can be easily implemented in hardware. One manufacturer I know of makes a Golay encoder/decoder in IC form: Space Research Technology (Houston, Tex., [713] 782-2244). This device is amenable to serial data streams of Golay encoded data. They also publish some application notes mathematically detailing the advantages of bit interleaving and the Golay code in general.

The references for this article describe various circuits suitable to the task of decoding-I mean suitable as a function of your personal Boolean logic overload level. The circuits are not trivial by any means, though the relatively small number of 4096 Golay codewords suggests that ROM-based

Hand verification of shift register calculation

```
                                     _______________________
110001110101 )10101010101000000000000
               110001110101

                110110111110
                110001110101

                   111001011000
                   110001110101

                     100010110100
                     110001110101

                       100110000010
                       110001110101

                         101111101110
                         110001110101

                           111100110110
                           110001110101

                             110100001100
                             110001110101

                               00101111001
```



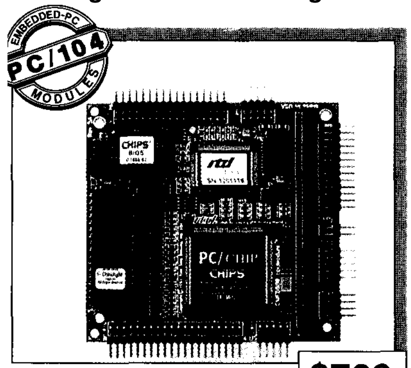
```
                          Shift register      Shift register
                              Bit 10 ─┐         ┌─ Bit 0
                                      │         │   ┌─ input data,
                                      │         │   │  LSB first
initial register value —>   00000000000         │
                    ┌─ 1  10101110001           1
                    │  2  11111001001           0
                    │  3  01111100100           1
                    │  4  00111110010           0
                    │  5  10110001000           1
value after shift ─ │  6  01011000100           0
                    │  7  10000010011           1
                    │  8  11101111000           0
                    │  9  11011001101           1
                    │ 10  11000010111           0
                    │ 11  01100001011           1
                    └- 12  10011110100           0
```

Figure 8—*Golay checkbit generator and example for polynomial* $x^{11}+x^{10}+x^6+x^5+x^4+x^2+1$.

decoders may be economical. However, the shift register implementations of the two Golay checkbit generators are simple, and are shown in Figures 7 and 8. As before, either circuit produces valid Golay checkbits. Also shown is a sample calculation for each. The final register value in Figure 7 is 11010000110, which we reverse and compare with the result of Figure 4, finding agreement. This circuit is equivalent to that calculation with input data of 555h. The final register value of Figure 8 is 10011110100,

**Replace Four Conventional PC/I 04 Modules with**
*One*
**CMF8680 cpuModule™**

**Embedded PC/XT Controller with Intelligent Power Management**



**PC/104 Compliant**
Actual Size: 3.6 x 3.8 x 0.6"

**$769**

- PC/XT compatibility with 286 emulation
- 14 MHz, 16-bit C&T F8680 CPU
- +5V only; 1.6W at 14.3 MHz, 1 W at 7.2 MHz
- Intelligent sleep modes, 0.1W in Suspend
- ROM-DOS and RTD enhanced BIOS
- Compatible with MS-DOS & real-time operating systems
- 1 M bootable solid state disk & free software
- 4K-bit configuration EEPROM (2K for user)
- 2M on-board DRAM
- IDE & floppy interfaces
- CGA CRT/LCD controller
- Two RS-232 ports, one RS-485 port
- Parallel, XT keyboard & speaker ports
- Optional X-Y keypad scanning/PCMCIA interface
- Watchdog timer & real-time clock

Expand This Or Any PC/104 System with the

**CM1 06 Super VGA Controller utilityModule™**

- Mono/color STN & TFT flat panel support
- Simultaneous CRT & LCD operation
- Resolution to 1024 x 768 pixels
- Displays up to 256 colors ($3951

Speed Product Development with the

**SK-CM1 06-X Starter Kit**

Your kit includes the CMF8680 cpuModule, CM106 SVGA controller, CM102 keypad scannina/PCMCIA utilityModule, CMF8680 cable-kit & VGA monitor cable for just $1295.

*Additional PC/104 compliant modules from RTD:*

- CM1 04 1.8" hard drive carrier utilityModule
- 12- & 14-bit analog I/O modules
- 12-bit, 4-20 mA analog output modules
- opto-22 & digital I/O modules

For more information on our PC/104 and ISA bus products, call today.

rtd®

**Real Time Devices USA**
200 Innovation Blvd. • P.O. Box 906
State College, PA 16804 USA
(814) 234-8087 / Fax: (814) 234-5218
**RTD Europa • RTD Scandinavia**
Real Time Devices is a founder of the PC/104 Consortium

#114

which agrees with the accompanying modulo-2 division remainder, when reversed. Also notice that, like the two polynomials, the shift register circuits are mirror images of each other, suggesting that some more gating could make one bidirectional shift register encode either polynomial.

## CONCLUSION

Of course, there are other error-correcting codes, but it is generally agreed in the literature that they are more difficult to correct than the Golay code, though the Hamming codes are easier. In general, the more capable the code, the more processor power it takes to decode. The Golay code could be used in even the smallest microcontrollers, such as the PIC series and 68HC05.

As a parting imperative, do your homework before choosing this or any other error detection/correction code. There are some important ideas that I did not discuss here, such as coding gain. This is an expression of the improvement in performance of the coded channel over the uncoded channel. Each code has a different coding gain for various channel noise conditions. You can also examine the references for matrix methods of encoding and decoding the Golay and other codes.

Finally, let me recommend the routine golay_test() in Listing 6. It does tests to verify the operation of your encoding and decoding routines by letting you induce a selectable number of errors in a test codeword on which correction is attempted. Always test your versions of these routines thoroughly before deploying; it is easy to slip up on a loop index and get a data-dependent error detector or corrector, which is certain to ruin a couple of nights' sleep. ❑

Listing **6**—*Golay [23,12] codeword test routine. This function tests the Golay routines for detection and correction of various patterns of error_limit bit errors. The error-mask cycles over all possible values, and error-limit selects the maximum number of induced errors.*

```
void golay_test(void)


  unsigned long
    error-mask,       /* bitwise mask for inducing errors */
    trashed_codeword,/* the codeword for trial correction */
    virgin-codeword; /* the original codeword without errors */

  unsigned char
    pass=1,           /* assume test passes */
    error_limit=3;    /* select number of induced bit errors here */

  int
    error-count;      /* receives number of errors corrected */

  virgin_codeword=golay(0x555); /* make a test codeword */

  if (parity(virgin_codeword))
    virgin_codeword^=0x8000001;

  for (error_mask=0; error_mask<0x8000001; error_mask++){
    /* filter the mask for the selected number of bit errors */
    if (weight(error_mask) <= error-limit) {
      /* you can make this faster! */
      trashed_codeword=virgin_codeword ^ error-mask;
      /* induce bit errors */
      decode(1, &error_count, &trashed_codeword);
      /* try to correct bit errors */
      if (trashed_codeword ^ virgin-codeword) {
        printf("Unable to correct %d error mask induced = 0x%lX\n",
              weight(error_mask), error-mask);
        pass = 0;


      if (kbhit()){   /* look for user input */
        if (getch() == 27)
          return;       /* escape exits */
        /* other key prints status */
        printf("Current test count = %ld of %ld\n", error-mask,
              0x8000001);



  printf("Golay test %s!\n", pass ? "PASSED" : "FAILED");
```

*Hank Wallace is the owner of Atlantic Quality Design Inc., an embedded systems hardware and software design firm located in Rural Hall, N.C. He may be reached at (910) 377-2843 or hwallace@cybernetics.com.*

## REFERENCES

1. Golay, Marcel J.E., "Notes on Digital Coding," *Proceedings of the IRE*, Vol. 37, June 1949, pp. 657.
2. Lin, Shu, *An Introduction to Error Correcting Codes.* Englewood Cliffs, N.J., Prentice-Hall, 1970, pp. 101–106.
3. MacWilliams, Sloane, *The Theory of Error Correcting Codes.* Amsterdam, North Holland Publishing Co., 1977, pp. 64-65, 482, 634-635.
4. Pless, Vera, *Introduction to the Theory of Error Correcting Codes.* John Wiley & Sons, 1982, pp. 1-13.
5. Shannon, C.E., "A Mathematical Theory of Communication," *Bell System Technical Journal,* Vol. *27,* July 1948, pp. 379-423; October 1948, pp. 623-656.

## IRS

404 Very Useful
405 Moderately Useful
406 Not Useful

# The Great Encryption Debate

FEATURE ARTICLE

**John Iovine**

Plaintext. Ciphertext. Clipper. There is a raging debate going on in cryptography circles these days, and everyone has an opinion. Find out some of the basics to be better prepared next time someone asks for yours.

Cryptology is a science of enciphering and deciphering messages and information. The word conjures up images of espionage, spies, hostile government action, and top secret information. We don't usually associate this word with privacy-your privacy-but it is this facet of cryptology that is being argued today in our courts and among government agencies.

## ENCRYPTED PRIVACY?

The arena where electronic bits of information are transmitted through data conduits is loosely termed "cyberspace." Currently, in cyberspace there's no guarantee of privacy. Transmitted messages may be intercepted and read indiscriminately. This possible invasion of privacy is not just limited to Email on your local BBS or on Internet. Our national telephone network, which handles voice and fax as well as computer telecommunication, is vulnerable. Additional data conduits like cable television systems and satellite feeds are becoming more commonplace all across the country. These newer networks are vulnerable to interception as well.

To better grasp the threat, imagine a company that routinely transmits bids or promotional information to field agents through one of these networks. The company can be put at a severe disadvantage if a competitor gains access to this information.

The dark side of our information age is that technically skilled crooks—sometimes romantically referred to as phreakers and crackers-can create havoc in your life. For a while, crackers were making national news by breaking into secured government databases.

Intercepting various unprotected data communications makes most people easy targets for others to gain access to confidential material. Anyone who has been electronically mugged has very little sympathy for these criminals. By stealing credit card numbers, they are capable of making purchases, charging telephone calls to your phone number, reading your Email, and listening to cellular phone conversations.

The problem is growing. Our national data network increases in size and complexity daily. It is changing and defining the methods by which people communicate, information is transferred, and business is conducted. It is therefore becoming increasingly important to secure the privacy of the networks and reduce their vulnerability to interception.

Business has been less than responsive to this threat. For instance, credit card companies justify their exorbitant +19% interest rates because they are needed to compensate for the tremendous amount of credit card (read "electronic") fraud and thievery. These companies should be doing much more to prevent electronic fraud instead of just passing the cost on to honest consumers in the way of high interest rates.

Rep. Edward J. Markey (D-Mass), the chairman of the House Telecommunication and Finance Subcommittee, had this to say about privacy: "Whether it's a cellular phone conversation, computer data, a fax transmission, a satellite feed, cable programming, or other electronic services, encryption is the key to protecting privacy and security." He stated further that "developing a national policy for encryption and its uses is therefore a process of fundamental importance for the future of our national networks and our competitive position internationally."

## ENTER THE CYPHERPUNKS

That's *cypher,* not cyber. Let's not confuse these similar sounding monikers. The cypherpunks want to

Listing 1-One *of the simplest forms of encryption is substitution. This piece of code prints out the 26 possible substitution alphabets.*

```
10 DIM plaintxt$(26)
20 FOR i = 0 TO 25
30 plaintxt$(i) = CHR$(65 + i)
40 NEXT i
50 FOR j = 0 TO 25
60   FOR i = 0 TO 25
70     PRINT plaintxt$((i+j) MOD 26);
80   NEXT i
90   PRINT
99 NEXT j
```

see widespread public use of crypto-technology. They see the individual's privacy protected through cryptography. However, they face powerful governmental and political obstacles.

The end of the cold war hasn't eliminated the need for cryptography and secret codes used by our government. But it should have alleviated some of the regulations concerning private use of cryptotechnology. The government still classifies crypto-technology with hard military weapons such as tanks. The U.S. government agency in charge of cryptotechnology is the National Security Agency (NSA). The cypherpunks see the NSA as trying to keep its monopoly on cryptotechnology intact.

One of the most outspoken and visible members of the cypherpunks is John Gillmore. Mr. Gillmore has this to say on the subject:

*Government investment leads to government control.
*Government control is detrimental to the development of the media.
*Government seized the control of radio and television in their infancy. Since then the media has never had full first amendment rights or protection.
•Encryption technology is the key for people and companies to maintain their privacy over the networks. The government should cease its involvement.

John has fought legal battles with the NSA on a few fronts. So far he has been victorious.

## BATTLE LINES

The lines are drawn. On one side you have the cypherpunks who feel that good public cryptographic technology safeguards our privacy. The NSA feels this is compromising our national security.

The government has threatened private cryptographers with jail. John Gillmore was threatened by the NSA stating that he was on the verge of violating the Espionage Act. A conviction would have sent him to jail for 10 years.

How can the government threaten private citizens? Easily: as stated previously, the government classifies cryptographic tools with military tanks and bomber planes.

## THE WASHINGTON CONNECTION

The Administration wants America to encrypt its information to protect it from unauthorized access. The encryption scheme, contained in the government-sponsored Clipper chip, includes voice as well as data information sent over communication lines. A major catch in this plan is that only the government-approved encryption is allowed in any device used by the government or in government projects. Other encryption methods continue to be legal for domestic use, but only in nongovernment applications.

The second catch is the potential for a trap door in the encryption chip's program that would allow law enforcement agencies to decipher any encrypted data. Therefore, this method of encryption doesn't alleviate concerns that the government could abuse its ability to tap into the privacy of the citizenship.

Of course, organized crime would use its own crytotechnology, anyway. So a trap door would only be effective for spying on small incidental crooks and private citizens.

The encryption algorithm touted by the Administration is contained in an integrated circuit. This chip, designed by Mykotronx in Torrance, Calif. and manufactured by VLSI in San Jose, Calif., is nicknamed "Clipper." It is a 12-Mbps encryption coprocessor. The OEM cost of the chip is $26 when purchased in large quantities, which trickles down to an increase of $100 in the street price of any electronic equipment (com-

Listing 2-*Going one step beyond Listing 1, fhis code first asks for a key letter, then generates a cipher alphabet initiated by the key letter.*

```
5 CLS
10 DIM PLAINTXT$(26), CIPHERTEXT$(26)
15 FOR I = 0 TO 25
20   PLAINTXT$(I) = CHR$(65+ I)
25 NEXT I
30 PRINT "Enter capital letter to code: ";
35 INPUT K$
40 FOR N = 0 TO 25
45   IFPLAINTXT$(N) = K$ THEN 60: REM code
50 NEXT N
55 CLS: PRINT "Error 1":END
60 REM code:
65 FOR J = 0 TO 25
70   CIPHERTEXT$(J) = PLAINTXT$((J+N) MOD 26)
75 NEXT J
80 PRINT: PRINT: PRINT "This is the cipher alphabet:"
85 PRINT
90 FOR I = 0 TO 25: PRINT CIPHERTXT$(I);: NEXT
95 END
```

puter, phone, fax] that contains the chip.

## SOFTWARE VS. HARDWARE

There are less expensive encryption chips on the market than the Clipper. Usually anyone interested in encryption takes a software approach. It may be a little slower than hardware, but the recurring cost is much less. Speed only becomes a critical consideration when it's necessary to secure fast communication such as video or voice communication.

## RECENT EVENTS

On February 4, 1994, the U.S. Government officially endorsed the Clipper chip and directed the Commerce Department's National Institute of Standards and Technology (NIST) and the Treasury Department to hold in escrow the keys used to unlock the Clipper codes. It also establishes new procedures for exporting products using Clipper to most countries.

The government has formed an interagency group whose job it is to develop encryption technologies that could serve as alternatives to Clipper.

The Clipper endorsement contains three flaws according to a policy paper released in January 1994 by the Institute of Electrical and Electronic Engineers: a classified algorithm, the key-escrow system, and an encryption standard developed for public use without public scrutiny.

The Clipper chip has developed many industrial and congressional opponents. So far, Novell, AT&T, Citicorp, Computer Associates, Hughes Aircraft, Motorola, and other major corporations openly oppose the Clipper encryption standard. The failure of recent administrations lies in the fact that they did not seek greater industry participation before proposing the Clipper chip. Further, they ignored protests from industry and Congress.

## THE BIG BROTHER ISSUE

The Clipper chip can provide government agencies with unprecedented wire tapping ability.

Ideally, the Clipper chip encrypts (scrambles) communication to everyone except the intended recipient. The key code to unscramble communication is held by two separate government agencies. The government has the option of using a joining key code to unscramble communications with court-approved legal authorization.

Listing 3—*Adding a bit more to the program in Listing 2, this program includes lower case and will encrypt a single line of text.*

```
5 CLS
10 DIM P1TXT$(26), C1TXT$(26),P2TXT$(26),C2TXT$(26)
15 FOR I = 0 TO 25
20   P1TXT$(I) = CHR$(65 + I): P2TXT$(I) = CHR$(97 + I)
25 NEXT I
30 PRINT "Enter letter key code:":
35 INPUT K$
40 FOR N = 0 TO 25
45 IF P1TXT$(N) = K$ THEN 65: REM code
50 IF P2TXT$(N) = K$ THEN 65: REM code
60 CLS: PRINT "Error 1": END
65 REM code:
70 FOR J = 0 TO 25
75   C1TXT$(J) = P1TXT$((J+N) MOD 26)
80   C2TXT$(J) = P2TXT$((J+N) MOD 26)
85 NEXT J
90 PRINT: PRINT: PRINT "This is the cipher alphabet:"
95 PRINT
.00 FOR I = 0 TO 25: PRINT C1TXT$(I);: NEXT
.05 PRINT
 10 FOR I = 0 TO 25: PRINT C2TXT$(I);: NEXT
 15 PRINT: PRINT "Enter a one-line message to encode ": PRINT
 20 INPUT TEXT$
 25 MLEN = LEN(TEXT$)
 30 FOR I = J TO MLEN
 35   FOR J = 0 TO 25
 40     IF MID$(TEXT$,I,1) = P1TXT$(J) THEN GOTO 155
 45     IF MID$(TEXT$,I,1) = P2TXT$(J) THEN GOTO 165
 50   NEXT J
155   MID$ TEXT$,I,1) = C1TXT$(J
160   GOTO 170
165   MID$ TEXT$,I,1) = C2TXT$(J
170 NEXT I
175 PRINT: PRINT "The enciphered message is:": PRINT  PRINT
180 PRINT  EXT$
185 END
```

However, there is a strong possibility that a trap door exists in the Clipper chip that would allow agencies unauthorized tapping. The government wouldn't allow the algorithm used in the Clipper, called "SkipJack," to be studied publicly, so no one knows for sure.

When the Administration endorsed the Clipper as a Federal Data Processing Standard on February 4, it was backed up with an immediate order for 50,000 Clipper chips. Meanwhile, a forced export embargo keeps all other encryption schemes expensive. U.S. manufacturers must "dumb down" their data encryption programs by keeping the key lengths to 40 bits or fewer for legal export. The Clipper uses an 80-bit code.

## ENCRYPTION BASICS

The following is a list of some of the basic terms that are used in encryption. *Plaintext* is the original unaltered message or file. *Ciphertext* is the encrypted message or file. An *encryption algorithm* is the function that maps plaintext into ciphertext. *Keys* are used to determine mapping. *Keyspace* describes the size of the key; it determines the number of all possible keys. For instance, an S-bit key has a keyspace of 256 (256 possible values), where a 16-bit key has a keyspace of 65,536. Keys are usually alphanumeric.

There are three main types of ciphers: *substitution, transposition,* and *product.* Substitution ciphers substitute each character in the plaintext with another, determined by the key. Transposition ciphers rearrange the characters in plaintext, again, determined by the key. Product ciphers combine the substitution and transposition algorithms.

A substitution cipher simply substitutes each plaintext character with another character determined by the key. For instance, we could easily displace the alphabet by one character to generate a simple substitution. For example, ABC.. .XYZ could become BCD.. .YZA, and the phrase "HELLO WORLD" would become "IFMMP XPSME." Substitution ciphers are also called Caesar ciphers, because Julius Caesar used this simple method of encoding messages.

The BASIC program in Listing 1 prints out the 26 possible substitution alphabets this system is capable of generating. Naturally, with a maximum of only 26 letter combinations, this system isn't very secure.

The program in Listing 2 is a little more sophisticated. It asks for a letter key code, then produces one cipher alphabet initiated by the key letter. Finally, the program in Listing 3 adds lower-case characters to the second program and allows the user to enter a one-line message for encryption.

The transposition cipher system rearranges the characters in plaintext. A simple system rearranges every two characters, so "ab" becomes "ba." With this kind of cipher, "HELLO WORLD" becomes "EHLLW ORODL."

## GENERATING MORE COMPLEX CIPHER SYSTEMS

Blaise de Vigenere, a French cryptographer in the sixteenth century, complicated the simple Caesar code. He proposed that the key be used to change the plaintext in a periodic manner. When a message is encoded by this method, you change a plaintext letter for each successive letter in the key, always running through the same sequence of key letters. A simple example should clear any confusion.

Suppose the name "John" was selected for the key code. This corresponds to the number sequence 9, 14, 7, 13. To encode a message using this key sequence, divide the letters of the plaintext message into groups of four. This corresponds to the four letters used in the key. To each letter group, add 9 to the number value of the first letter of each group, 14 to the second letter, 7 to the third letter, and 13 to the fourth letter. The example below illustrates the Vigenere code:

```
Key Code: JohnJohnJohnJohn
Plaintext message: helloworld
Ciphertext message: qssy xlvf m
```

As you can see, the coding algorithms are becoming more complex. Even this code pales to the more sophisticated programs available.

## THE DEBATE CONTINUES

I've only scratched the surface in the great encryption debate. There are a number of on-line newsletters carried on the Circuit Cellar BBS that follow the issue closely (Computer Underground Digest [CuD] and Electronic Frontier Foundation [EFF]). If you are interested in following along, check them out.

So what do you think? Write and let me nkwo (pun intended). ❑

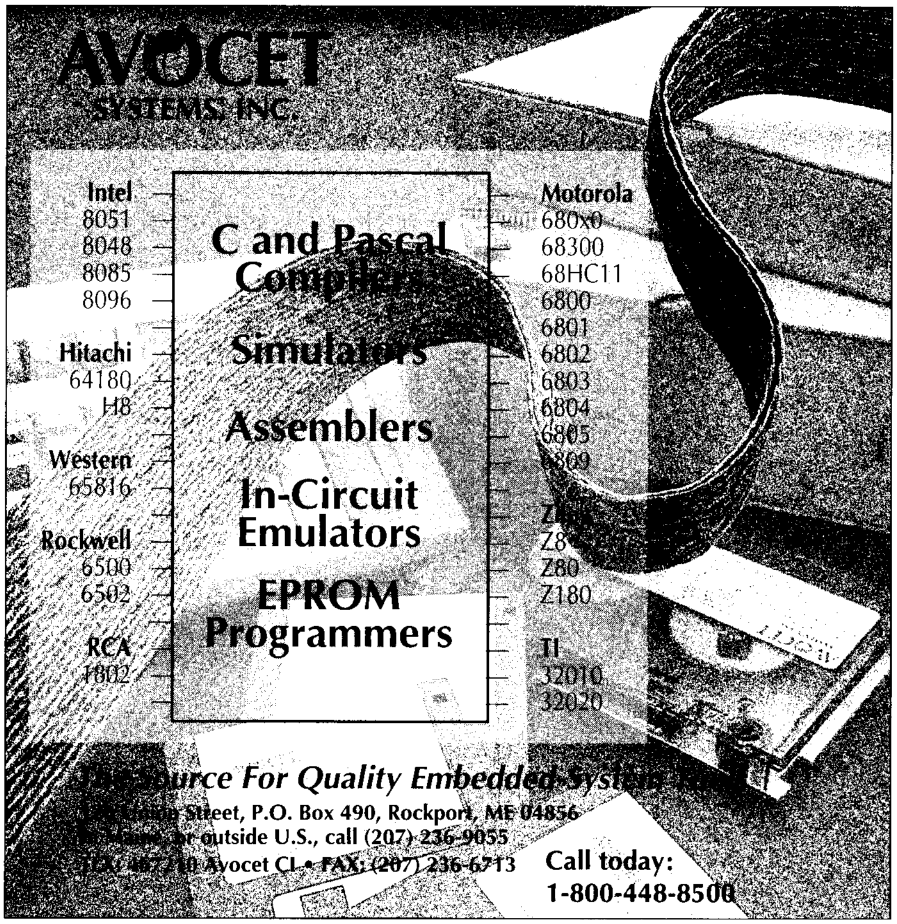
*John Iovine is a free-lance writer living in Staten Island, N.Y. He has published numerous books on electronics- and science-related topics. He may be reached at 75425.673@compuserve.corn.*

## CONTACT

For those who wish to pursue data encryption, Images Company offers an encryption program titled Cipher 1 .O for $9.95. Images Company, P.O. Box 140742, Staten Island, NY 10314, (718) 698-8305. New York residents must add the appropriate sales tax. Add $5.00 postage and handling to all orders.

## I R S

407 Very Useful
408 Moderately Useful
409 Not Useful



#116

# Take a Bus to the Nearest Star

FEATURE ARTICLE

**Michael Swartzendruber**

## The Star-485 Adds Flexibility to any RS-485 Network

Daisy chained connections and under 32 nodes work just fine for many RS-485 applications. However, what happens when you need to go beyond the spec? Add a Star-485 or a BR-485 and go beyond the limits.

One of the most enticing features of the RS-485 networking standard is its low cost. Its simple method for connecting new nodes to the network is another reason the standard is so widely adopted.

This article presents some different options that can be applied to RS-485 networks to more effectively manage its topology. One of these devices is an in-line signal regenerator for RS-485 networks. The second is a simple low-cost bridge/repeater for RS-485 networks. And the third is an RS-485 hub that can be used to create a star network out of an RS-485 network. All of the prototypes for these devices are shown in Photo 1.

### TWO WIRES ARE BETTER THAN ONE

The increasing application of controllers for factory, building, automobile, and home automation means more controllers are sprinkled across any typical installation. It is a natural progression-from a systems standpoint-to connect these controllers together forming a centrally managed (or monitored) distributed control system.

A distributed control system implies data and/or command transmissions between nodes. This creates an instant demand for reliable data transmissions between nodes. Since nodes may be distributed over large distances, and because they probably exist in electrically hostile environments, the method of signal transmission between nodes *must be* reliable and robust if the system is going to perform meaningful work.

Several factors contribute to the reliability of copper-wire data links. Restricting the discussion to properties of the physical layer (ignoring error detecting and correcting protocols) allows the focus to remain on maximizing the immunity of the physical layer to environmentally generated electrical interference.

Of the interconnection methods that could be considered (single wire, double wire, ribbon cable, twisted pair, shielded twisted pair, coaxial, fiber optic), unshielded twisted-pair connections present a good compromise. First, the impedance of the connection is controlled and uniform over the length of the cable. This reduces problems of signal aberrations that appear when a signal is subjected to cable impedance discontinuities. In addition, the induced noise on the signal from the environment is reduced by mutual coupling between the signal-carrying wires since they lie in the same signal plane over the length of the cable. Shielded twisted pairs, fiber optics, and coaxial cable can provide additional noise margins, but the additional cost of the cable preparation and the higher cost of hardware offsets their advantage.

A powerful way to enhance the strengths of twisted pair cabling is to propagate balanced signals (i.e., differential signals). The advantage of this transmission method is increased noise immunity due to the common mode rejection characteristics of differential receivers. Any steady state or sporadic noise induced onto the cables from the environment will affect each conductor to an equal degree (since they are in the same noise plane], and will be rejected as common mode noise by the receivers.

As the strengths of differential signaling systems began to become an

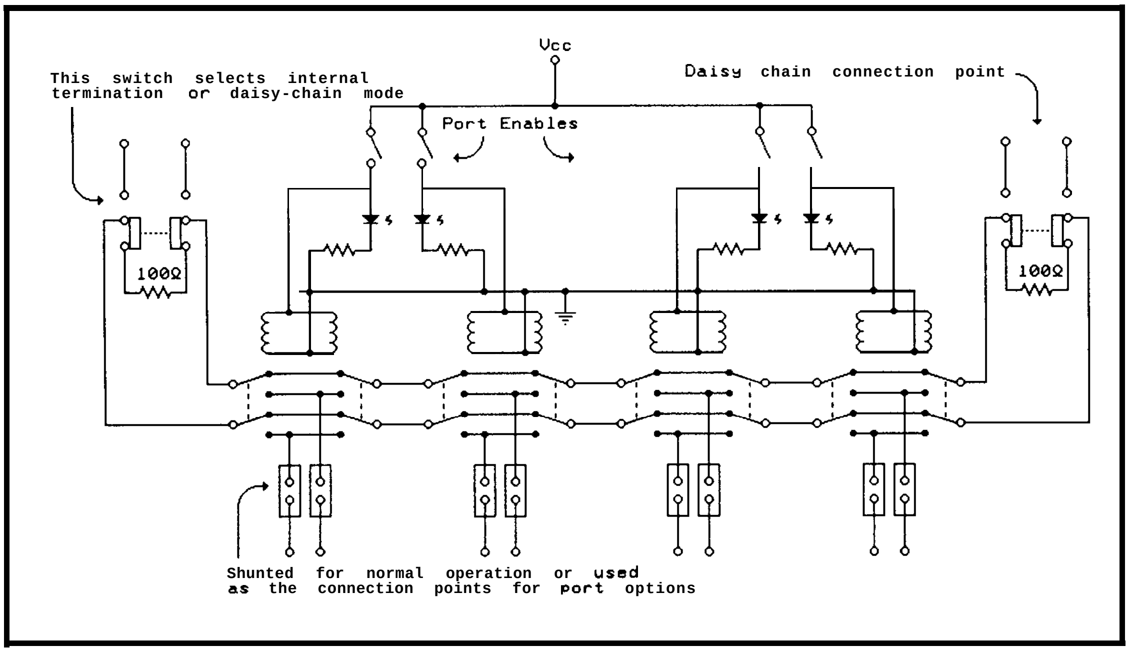

Vcc
This switch selects internal termination or daisy-chain mode
Daisy chain connection point
Port Enables
100Ω
100Ω
Shunted for normal operation or used as the connection points for port options

Figure 1--The *Star-485 is a simple switching arrangement to extend RS-485 networks and isolate portions of the net.* Semiconductor *enthusiasts can subsfifufe silicon-based switches to rep/ace the relays if desired.*

accepted solution for data transmission, the industry recognized the need for a standard. EIA standard RS-422-A defines a differential system that supports a single driver and up to ten receivers on the "bus." This specification defines the maximum end-to-end distance of the cable as 4000 feet and also defines the performance parameters of receivers and transmitters.

The EIA RS-422-A standard was improved with the RS-485 standard in 1983. This standard allows up to 32 receivers/transceivers to reside on a single bus. Since there can be multiple driver/receiver pairs present on the bus, the devices must be designed to handle that situation. An RS-485 driver must provide internal self-protection against the possibility of contention, which is defined as the state when more than one transmitter is attempting to drive the bus.

The standard specifies no more than 32 ***unit loads*** (a unit load is defined as a single driver/receiver pair) be present on the bus. In a similar manner, the physical geography of the network must not be exceeded because '485 drivers are designed to operate with a characteristic load presented by the maximum allowable cable length specified by the standard.

## THAT'S GREAT! ISN'T IT?

The shortcomings of this networking method are related to the fact that RS-485 uses a bus-oriented connection method. While the bus topology helps keep costs low, it can be a great detriment in certain conditions.

One weakness occurs when a cable break occurs somewhere in the bus, effectively killing the network. The network ceases to function because it is broken into two incomplete networks, and nodes on different sides of the break can no longer communicate with one another.

Depending on the geographic size of the network and the accessibility of the network segments (the cable installed between adjacent nodes), troubleshooting this kind of failure can take a significant amount of time.

The second case where a bus topology falls short is in the ability to troubleshoot network nodes. In the case where one or more devices on the network are failing (jabbering or contributing to network collisions), each node on the network must be investigated to determine if it was one of the failing elements. You might have to power down each node or remove its network connections and then determine if the problem disappears. This must be done for each station on the network until the problem is found.

This process can also be time consuming because each node must be visited. The best you can hope for is that each node is easily accessible. The fact that the network is down for the

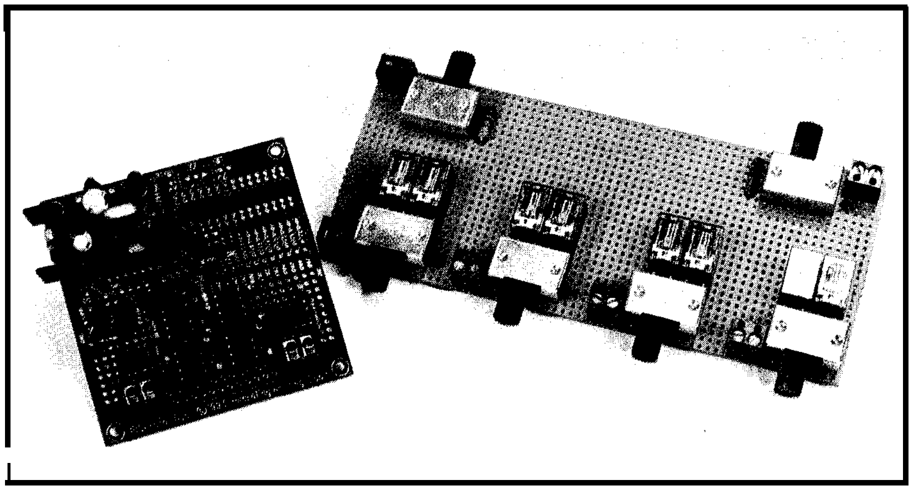


Photo 1—*The BR-485 on the left and the Star-485 on the right provide the means for extending and configuring RS-485 nefworks.*

AMX™

The Real-Time Multitasking Kernel

680x0, 683xx
80x86/88 real mode
80386 protected mode
i960® family
R3000, LR330x0
Z80, HD64180

**Features**

- Full-featured, compact ROMable kernel with fast interrupt response
- Preemptive, priority based task scheduler with optional time slicing
- Mailbox, semaphore, resource, event, list, buffer and memory managers
- Configuration Builder utility eases system construction
- InSight™ Debug Tool is available to view system internals and gather task execution statistics
- Supports inexpensive PC-hosted development tools
- Comprehensive, crystal clear documentation
- No-hidden-charges site license
- Source code included
- Reliability field-proven since 1980

**Count on KADAK.**
Setting real-time standards *since* J 978.

For a *free* Demo *Disk* and your copy *of our* excellent *AMX product description*, contact *us today*.
Phone : (604) 734-2 796
Fax: (604) 734-8114

**KADAK Products Ltd.**
*206 - 1847* West Broadway
Vancouver, BC, Canada V6J1Y5

AMX is a trademark of KADAK Products Ltd. All trademarked names are the property of their respective owners.

#117

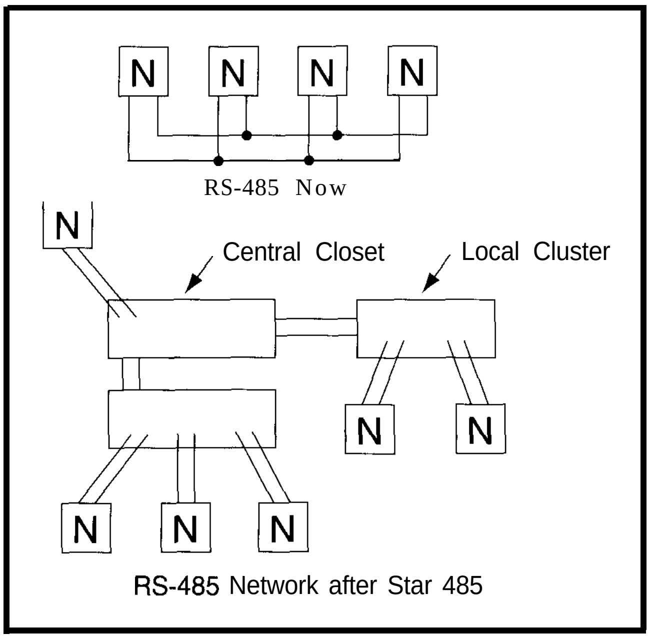


Figure 2—*Traditional RS-485 networks are severely limited in how they may be wired. Using the Star-485 expands the possibilities.*

duration of the troubleshooting session can have a major impact on those sites where the RS-485 network is used for mission-critical tasks.

Another area where the bus topology falls a short is in adding new nodes to the network. If you need to place a new node on the net, where does it go? Does it get tacked on to the end of the network! You can either extend the network to reach this new node or you can break the network at the closest segment and install the new node in the segment between two previously existing nodes. Hopefully, your mental map or cable management diagrams are accurate. In any case, network additions can be somewhat troublesome.

## LEARNING BY EXAMPLE

The Ethernet community made wide use of bus architectures in the early evolution of this networking standard. As these networks grew, the effort required to maintain, expand, and repair them became more problematic. The Ethernet community realized that these failures could cause a network to be pitched right out the of building, so they addressed the issue by providing for a new topology. They modified the standard to allow for a *star* topology.

The star topology provides many benefits to the persons who have to manage or repair networks. A key benefit provided by the star topology is that it simplifies troubleshooting the network. Other benefits of the star are that it eases cable management and network expansion.

## WATCH ME GROW

What do networks do? Well, mostly three things. The first is they move data around-day in and day out. The second is they break. The third is they grow, and they grow, and they grow.

There are two fundamental ways that networks grow. One is in number of devices attached to the network. The other is they get larger in a geographic sense. In either case, you will eventually push the limits of the network's specifications (number of devices or maximum allowable size). But you may be able to apply one of the devices presented here to get you out of the jamb.

## ENTER THE STAR-485

The Star-485 allows an RS-485 network to be configured as a star. The method used to perform this is to "can the bus," meaning the bus is contained in a modular unit. Nodes join the network by connecting to ports on the Star-485. This connection method creates a star.

The schematic for the BR-485 is shown in Figure 1. What happens at the ports defines the way the Star-485

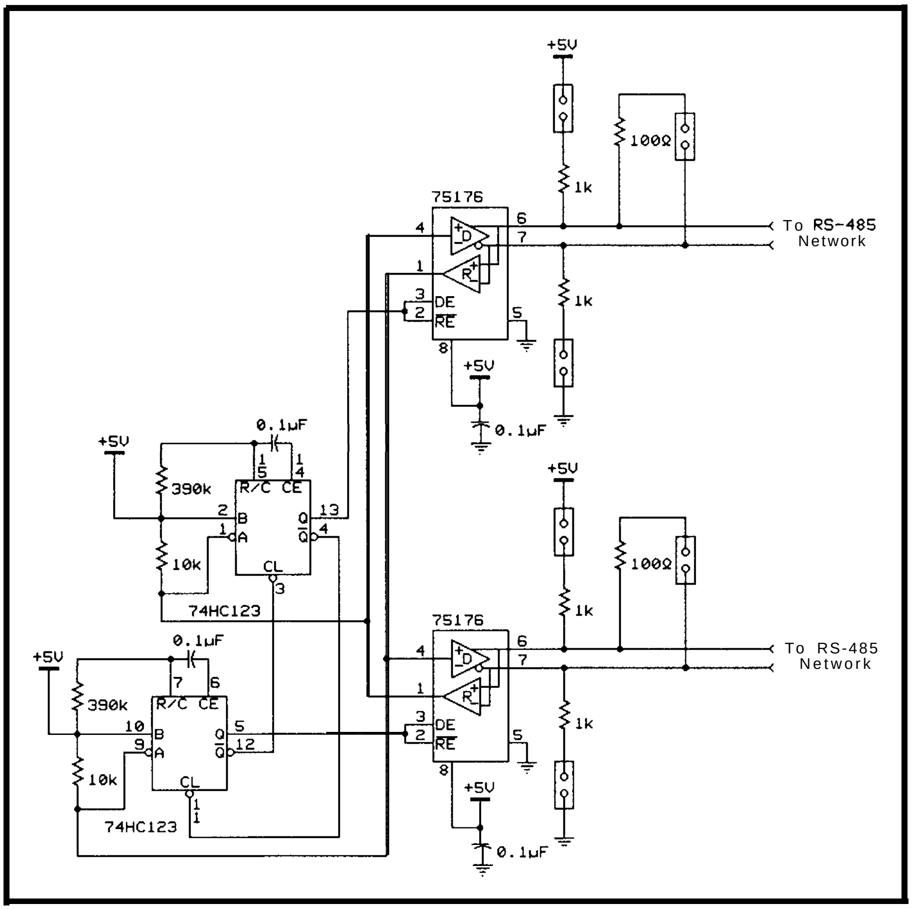


Figure 3-An adaptation of *a circuit originally designed by Mitchell Lee that appeared in the February 1994 issue of* Linear Technology *magazine, this version is designed to operate at 9600 bps and is published here in its adapted form with fheir permission.*

works. When the port is disabled, the network bus bypasses the connection to the world outside that port. This isolates the port from whatever is connected at that port, and likewise isolates the network from whatever is connected at that port. When the port is enabled, the path the bus takes past that port is diverted and the port (and whatever is connected to that port) is now a member of the bus. The two termination resistors at either end of the bus are there to complete the LAN in the can, and they can be switched out to extend the network beyond the confines of a single hub.

Each node home-runs back to a central location. This location provides an easy way to test and identify each network segment. If a network

▶ Project Parts ◀

| | |
|---|---|
| **New Stuff** | |
| PCF8574 I2C Remote 8-bit I/O port | 9.00 |
| DS1620 Remote digital thermometer/thermostat | 10.20 |
| SAA1064 I2C 4-digit LED driver | 12.30 |
| BQ2010 NiCd/NiMH battery gas gauge | 19.25 |
| AD75019 Analog 16x16 MUX w/PLCC socket | 41.25 |
| MAX456/457 Analog 8x8 MUX and video drivers | 75.80 |
| **Firmware Flyers (postpaid in US / with order)** | |
| 8051 Firmware Debugging Techniques (65 pg w/ 3.5" disk) | $18 / 15.00 |
| Introduction to the 8051 Instruction Set (46 pages) | $10 / 8.00 |
| '386SX Project: FDB schematics (17 pages or so) | $8 / 5.00 |
| 8051 Instruction Reference Card | $3 / 2.00 |
| 8255 Cheat Sheet Reference Card | $3 / 2.00 |
| **Stepper Motor & Power Drivers** | |
| ULN3751Z Power op amp (±3V to ±3V supply, 3.5 A output) | 4 60 |
| UDN2993B Dual full H-bridge bipolar stepper motor driver | 6.30 |
| UCN5841A Serial-input, 8 latched 500 mA sink drivers | 6.90 |
| UCN5895A Serial-input, 8 latched 250 mA source drivers | 7.50 |
| UCN5804B Unipolar stepper motor translator/driver | 8.00 |
| UDN2998W Dual full-bridge stepper driver: 50 V, 2 A | 13.80 |
| **IR Remote Control** | |
| LD273 dual IR LED (bright, wide beam) | 2.10 |
| IR3C02A laser control (±5V: LT022MC/LN9705P) | 2.30 |
| IR3C07 laser control (+5V: LT022PD/LN9705) | 2.85 |
| PH302 fast IR photodiode. | 3.10 |
| GP1U52Y 40 kHz IR receiver (side-looking) | 4.00 |

| | |
|---|---|
| IS1U60 38 kHz IR receiver. | 4.80 |
| IRSAMPLE parts (PH302, LM311, etc. MCIR-Link, INK 29) | 5.70 |
| Excellent IR filter (opaque to visible light, 35 mm slide mount) | 6.75 |
| MC145030 IR encoder/decoder | 6.75 |
| IR-Link I/O: IS1U60, LD273, CD4047, schematic | 10.40 |
| **Other Neat Stuff** | |
| UGN3503U Ratiometric linear Hall Effect sensor | 2.25 |
| (39 page Hall Effect app notes with '3503 order only) | 3.90 |
| DS1232 Micromonitor watchdog | 6.75 |
| CS212 S-ART I/O network/security monitor | 8.10 |
| DS2400 Silicon Serial Number (2 chips) | 8.40 |
| TDA8444 I2C bus octal 6-bit DAC | 8.75 |
| DS1210 NVRAM power conroller | 9.75 |
| DS1231 Power monitor...... | 9.90 |
| DS1202 Serial clock/calendar w/crystal | 10.20 |
| PCF8583 I2C bus clock/calendar w/crystal | 10.70 |
| MT8888 DTMF Transceiver (Intel bus interface) | 10.50 |
| MAX691 Power supervisor.. | 11.85 |
| 75T204 DTMF decoder...... | 12.00 |
| PCF8591 I2C bus quad 8-bit ADC, single 8-bit DAC | 14.10 |
| MAX233 self-contained +5V powered RS-232 interface | 14.25 |
| BQ2003 Fast-charge NiCd/NiMH battery controller | 14.55 |
| MT8808 8x8 bipolar signal analog crosspoint | 16.10 |
| HBCS-1100 High-resolution HP bar code sensor | 32.80 |
| Firmware Dev Board ICs, schematic thru #36 | 45.70 |
| MAX252 opto-isolated +5V RS-232 interface | 64.00 |
| CH1840 FCC pre-approved full-function DAA | 72.50 |

**...and much more...**

UPS Ground/2nd day $6.50/9 to 48 US states, COD add $4.50. PO Boxes and Canadian orders $6 for USPS mail. Check, MO, or COD only, no credit cards. POs add $50, call first. NC residents add 6% sales tax. Quantity discounts start at five parts. Data sheets included with all parts.

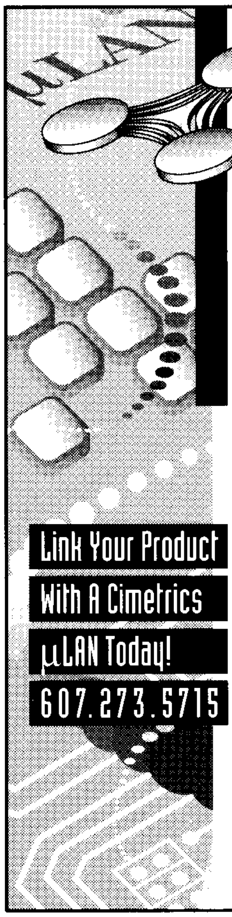
Call/write/fax for *seriously* tempting catalog...

**Pure Unobtainium**

▶ Your unusual parts source ◀

13109 Old Creedmoor Road • Raleigh, NC 27613

FAX/voice (919) 676-4525



Link Your Product
With A Cimetrics
µLAN Today!
607.273.5715

*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.

- 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

120 West State Street • Ithaca, New York 14850
Ph 607.273.5715 • Fx 607.273.5712

#118

segment fails, it is easy to test the entire physical layer of the network from a single location. The centralized location provides better options for cable troubleshooting and also provides for better cable management. Instead of having the network snaking all around the place, it all comes back to the Star-485, where each segment can be clearly marked and identified.

Each network node can be switched into or out of the network, without having to remove a cable connection, by a port-enable switch for each port. This speeds up the test and isolation of nodes that may be causing network errors. To remove a node from the network, disable the port the node is connected to. When the port is enabled, the node is part of the network again. This design simplifies the task of node removal and insertion by selectively disabling and enabling ports. This makes the divide-and-conquer approach to network repair much easier. Figure 2 shows the topologies possible with the Star-485.

## HANDLING EXPANSION

Adding a new node to the network couldn't be easier. Just run the cable between the device and the Star-485 hub and enable the port. Note that polarity rules still apply, so make certain the positive and negative signal lines are connected correctly at each end.

Each Star-485 hub can be chained to additional hubs. This allows simple expansion of the network by "chaining" hubs to one another. Connecting one hub to the next effectively daisy chains the hubs. In order to enable the chained connection, the terminator on the chained port must be deselected. When the terminator is disconnected (by a slide switch), a cable installed between hubs joins both hubs in a single network.

The hubs may be located next to each other for a single centralized wiring closet or they can be distributed in a number of wiring closets supporting clusters of devices in different locations. This option connects nodes to a hub that is closer to their location and then runs a single connection between hubs.

For troubleshooting purposes, hubs can be disconnected from one another by reselecting the terminators in each hub. This breaks the connection between the hubs and creates isolated subnets in the original network. Remember to select the terminators at each hub or else the subnet connected through the unterminated hub may fail because of a lack of termination.

## PORT OPTIONS

To keep entry costs low, the Star-485 can be configured as a completely passive switch. The hub merely provides the connection point for network nodes to the bus in the hub.

Signal regeneration may be needed for nodes that are located some distance from the hub or when the segment connecting the node to the hub runs by some relatively noisy radiation sources. To provide for this, there are couple of designs that can be installed in the segment at any one (or all) of ports of the hub on a port-by-port basis. These circuits are applied to only the ports that require the service.

This first design is a signal regeneration amplifier. Its purpose is to improve the signal quality of the network. Instead of just passing the signal straight to the node, the signal is passed through a nonlatching,

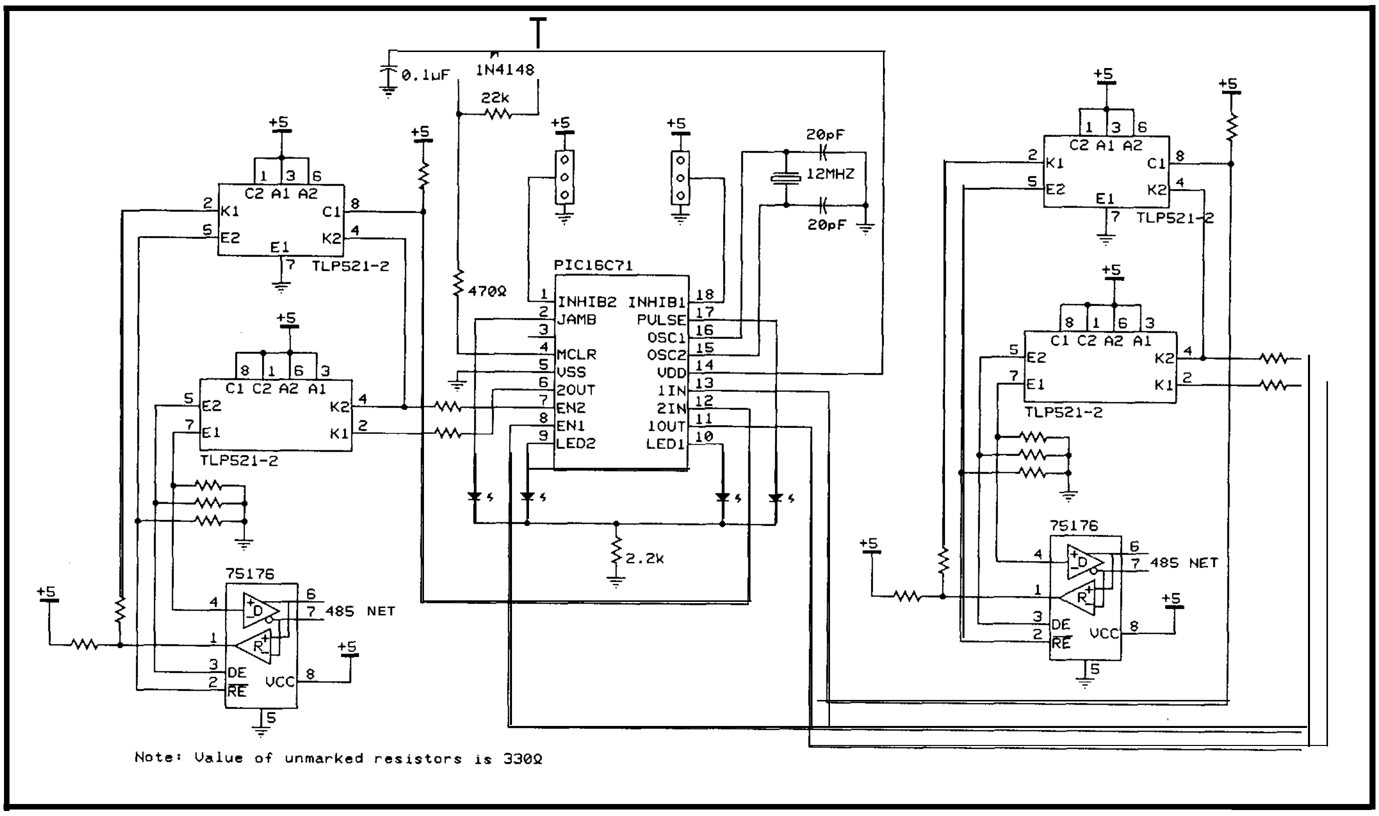


Figure 4—*The BR-485 circuit is little more than a PIC with some interface chips and some LEDs that serve as network traffic indicators.*

noninverting regenerative circuit This allows any node to be located a considerable distance from the hub. The schematic for the circuit that performs this signal regeneration task is shown in Figure 3.

This circuit is "tuned" to operate at 9600 bps. It operates by monitoring both of the RS-485 networks for a start bit. When a start bit is seen on one side or the other, the one-shot is fired which switches the opposite network (relative to the incoming start bit] into transmission mode. The duration of the transmitter's on time is about 10 bit times, at which point the one shot expires and the circuit resets itself.

## THE "BR-485" OPTION FOR THE STAR-485

For those who want to completely wild, you can choose to plop in the BR-485 (the term BR-485 is an acronym for Bridge Repeater 485). Those of you familiar with devices commonly found in high-speed data networks are probably already aware of the general function of a bridge repeater. But for the benefit of those of you not familiar with them, a short explanation follows.

A bridge repeater is a simultaneous member of two or more physically separate networks. The simple bridge repeater typically will not generate traffic of its own. Its sole purpose is to listen to each network, and whenever it senses traffic on any network, it immediately forwards that data to all other networks.

That said, the BR-485 is a simultaneous member of two physically separate RS-485 networks and can be used to extend the physical dimensions of an RS-485 network beyond the bounds specified by the standard. It does this by bridging the network segments together into a single *logical* network. Because the network still consists of physically isolated segments, the capacitive/loading effects that would be brought about by a direct connection between the network segments can be ignored and is therefore not a consideration. Also, both of the network segments can grow to the physical size specified by the standard without any overall adverse effects.

## HOW IT DOES IT

When the BR-485 is initialized, it listens to both of the networks for any traffic. If any traffic is identified, the BR-485 checks the other network for traffic. If traffic is found to be present on both networks, then a collision is taking place. In this event, no traffic is forwarded to either network and the "Collision LED" will light to indicate the condition. In the event no collision is detected, the traffic from one network is immediately forwarded to the other network. Whenever traffic is being forwarded through the BR-485, a "Net Bit Forwarded" LED will light up for the duration of the packet. This provides a visual indication of which network is generating the traffic.

If you look at Figure 4, it's pretty apparent that a single-chip controller (a PIC) runs the show in the BR-485. The rest of the components in this device support the I/O interfaces for the controller or are provided for the benefit of the user. The flowchart for the program running in this PIC is shown in Figure 5.

The pair of 75 176s are used as the network I/O ports. The LEDs connected to the controller are used to provide an indication of what the device is doing. One of them (connected to the pulse output) is the heartbeat of the system. It toggles every time the BR-485 code goes through its idle loop. Don't count on seeing it blink, though, because this processor is pretty zippy. The LED

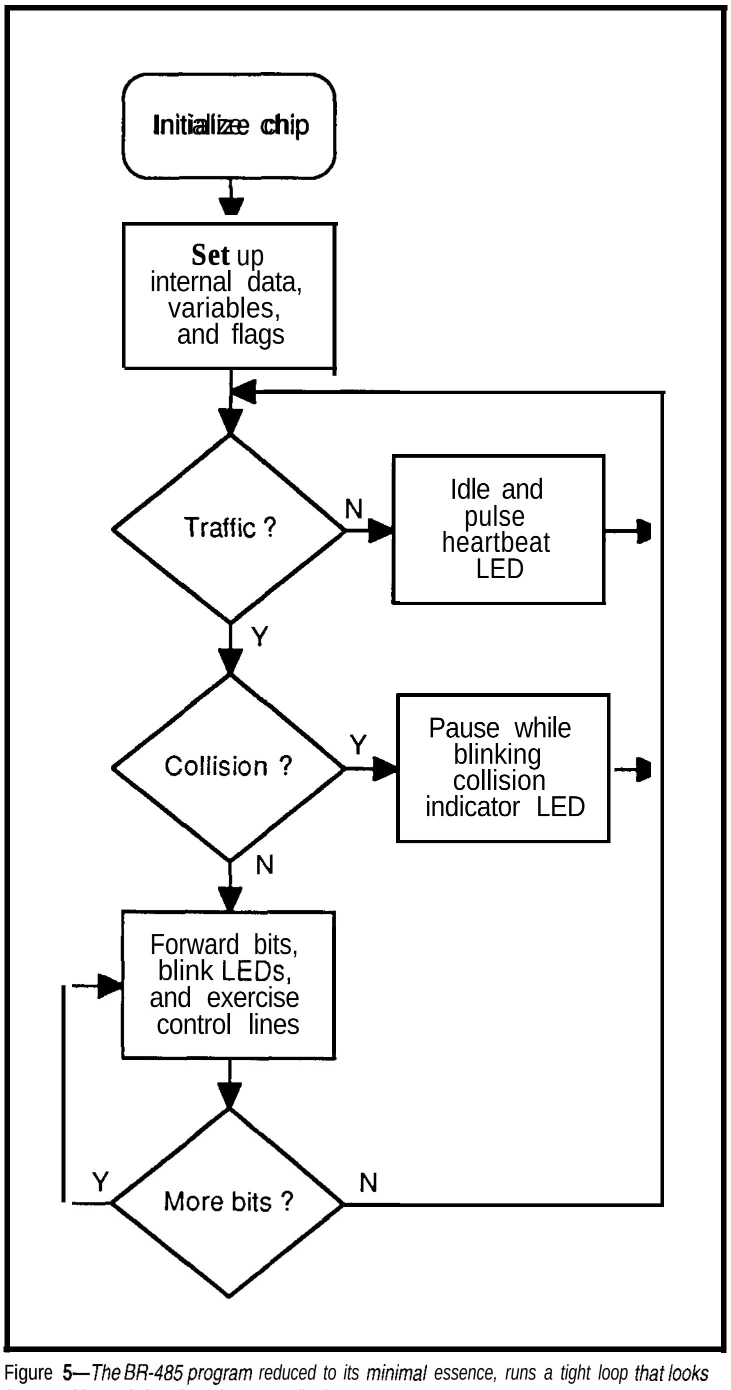

Initialize chip
Set up internal data, variables, and flags
Traffic ?
N
Idle and pulse heartbeat LED
Y
Collision ?
Y
Pause while blinking collision indicator LED
N
Forward bits, blink LEDs, and exercise control lines
Y
More bits ?
N

Figure 5—*The BR-485 program reduced to its minimal essence, runs a tight loop that looks for start bits and then branches accordingly.*

indicators blink on for the duration of a transmission. It is normal for these two LEDs to flash on and off during normal operation. The fourth LED, connected the JAMB output, blinks on for a short time if a collision is sensed. The small handful of resistors are acting as either pull-ups or current limiters.

## NETWORK SECURITY AND TRAFFIC CONTROL

The BR-485 contains a 2x3 header that can be used to control the forwarding behavior of the BR-485. For full-duplex operation, both of the center pins on the header will be shunted to ground. If the center pin on one of these headers is shunted to 5 V, then the network controlled by that pin will not be monitored for traffic. This creates a simplex repeater and traffic will only flow through the BR-485 in a single direction.

If both of the center pins are shunted to 5 V, then the BR-485 enters a "blocked" state. As long as the BR-485 is blocked, no traffic will be forwarded through the device, regardless of which network generates traffic. The blocking feature can be useful for controlling access to the network. If the center pins of these headers are brought under the control of an external controller, then you have the basis of a controlled network access system.

## TYING IT ALL TOGETHER

The Star-485 serves a very useful purpose in making RS-485 networks easier to test, maintain, add to, and manage. If you agree that the bus topology has weaknesses and would prefer to install a star topology for your RS-485 network, then the Star-485 gives you that option. Since it uses a modular design, its initial entry costs are very low but it can be expanded to support unlimited stations. Since the hubs do not have to be located in a single location, it is possible to create distributed stars that would serve local clusters of devices. Larger systems can also be easily subnetted and the network load can be distributed quite easily by connecting nodes from one hub to another.

Now, not every RS-485 network can justify the application of a star topology. But the idea has great merit in those cases where the network is large, contains many devices, or is running a mission-critical application. The reduction in down time is the paramount concern in these cases and the star helps to lessen the time the network is down by providing an architectural advantage.

The simple RS-485 signal regenerator is a low-cost way to get some additional noise or distance margins in an RS-485 network. It can be easily spliced into a line wherever a signal boost is needed.

The BR-485 can be used to solve a wide variety of common network problems. The simplex and full-blocking modes can also be applied to control the access to your network or it can be used to control the way data flows in your network. When you consider this device in the big scheme of things, it may not be a space shuttle, but it is just about the smartest piece of wire I have seen yet. ❑

*Michael Swartzendruber is an engineer with experience in network and communications design and Windows and Macintosh programming. He may be reached at 75110.3302@compuserve corn.*

## SOURCE

Preprogrammed PICs for the BR-485 are available. Contact Michael through CompuServe for more information.

## 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

410 Very Useful
411 Moderately Useful
412 Not Useful

**Real-Time Multitasking with DOS**

**for Microsoft C, Borland C, Borland/Turbo Pascal**

Develop Real-Time Multitasking Applications under MS-DOS with ***RTKernel!***

RTKernel is a professional, high-performance real-time multitasking kernel. It runs under MS-DOS or in ROM and supports Microsoft C, Borland Ctt, Borland/Turbo Pascal, and Stony Brook Pascal+. RTKernel is a library you can link to your application. It lets you run several C functions or Pascal procedures as parallel tasks. RTKernel offers the following advanced features:

- preemptive, event/interrupt-driven scheduling
- number of tasks only limited by available RAM
- task-switch time of approx. 6 μs (33-MHz 486)
- performance is Independent of the number of tasks
- use up to 64 priorities to control your tasks
- priorities changeable at run-time
- time-slicing can be activated
- programmable timer interrupt rate (0.1 to 55 ms)
- high-resolution timer for time measurement (1 μs)
- activate or suspend tasks out of interrupt handlers
- programmable interrupt priorities
- semaphores, mailboxes, and message-passing
- keyboard, hard disk, and floppy disk idle times usable by other tasks
- interrupt handlers for keyboard, COM ports, and network interrupts included with source code
- supports up to 36 COM ports (DigiBoard, Hostess boards)
- supports protocols XOn/XOff, DTR/DSR, RTS/CTS
- full support of NS16550 UART chip
- supports math coprocessor and emulator
- fast, inter-networkcommunication using Novell's IPX
- runs under MS-DOS 3.0 to 6.x, DR-DOS, LANs, or without operating system
- DOS calls from several tasks without re-entrance problems
- supports resident multi-tasking applications (TSRs)
- runs Windows or DOS Extenders as a task
- supports CodeView and Turbo Debugger
- Kernel Tracer for easy debugging
- ROMable
- full source code available
- no run-time royalties
- free technical support by phone or fax

**RTKernel-C** 4.0 $495
C Source Code: add $445

**RTKernel-Pascal 4.0** $445
Pascal Source Code: add $375

International orders: add $30 shipping and handling.
Mastercard, Visa, check, bank transfer accepted.

On Time
MARKETING
Professional Programming Tools

In North America, please contact:
On Time Marketing
88 Christian Avenue • Setauket, NY 11733 • USA
Phone (516) 689-6654 • Fax (516) 689-l 172
CompuServe 73313.3177

Outside **North** America, please contact:
On Time Marketing
Karolinenstrasse 32 • 20357 Hamburg • GERMANY
Phone +49 - 40 - 43 74 72 • Fax +49 - 40 - 43 51 96
CompuServe 100140,633

#121

## DEPARTMENTS

## FIRMWARE FURNACE

Ed Nisley

# Journey to the Protected Land: The '386SX Takes the First Step

Protected-mode programming is not for the faint of heart, especially in embedded applications. However, it can be a useful skill to master, so let Ed be your guide as he heads into this sometimes frightening land.

So there you are, eyeing the offerings at Ralph Rotten's Spoilsport Motors. You spot a new car with serious V-8 mojo, antilock brakes clamping fat radials, and surround sound you can feel. Dirt bounces off the paint, the interior has short-chain hydrocarbon zing, and the alarm system launches on warning.

Who cares if four cylinders don't fire, the ABS locks up sporadically, the sound system is full of tangled wiring labeled DANGER in eight languages, and the alarm system keeps *you* out? It's fast and that's what counts....

Such is the state-of-the-art in real-mode PC programming, with the possible exception that code doesn't smell quite so nice. At every turn you're working against the hardware: not only do you find hairpin switchbacks along the way, but the CPU wants to go straight. Fast, yes, but you have to like driving off the edge now and again.

There is a better way. Now that we've stuffed the Firmware Development Board full of hardware that will help us see what we're doing, it's time to return to firmware. This time, though, we're going for code that fires on all cylinders....

## WHITHER THE PROTECTED LAND?

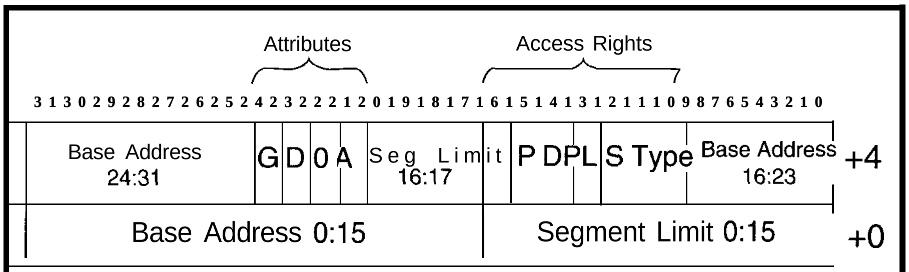
With the advent of the Intel 80386EX embedded processor, this “embedded PC” stuff suddenly starts looking positively mainstream. Your stock PC clone is now a bond-out CPU emulator: all the techniques I’ve been discussing are applicable to 386EX programming. Those PC compatibility barnacles have enough sticking power to determine what the latest silicon must look like: if it’s not precisely like a PC, it just doesn’t survive.

The catch: we’ve been using 16 bits of a 32-bit CPU, limiting it to a megabyte of memory address space, and giving up lots of debugging assistance. It’s a shame, but the barnacles work against us as well as the hardware folks, too.

Using the full power of the ‘386 CPU (and the ‘486 and the Pentium) requires leaving the familiar confines of real mode and entering the uncharted territory of protected mode. Suddenly segments can span 4 gigabytes, you can’t overwrite your code (unless you try real hard), and the CPU keeps you from stomping on data at the end of a null pointer. It all sounds interesting until you learn that FAR JMP and CALL instructions can run for hundreds of clock cycles. Well, there are always tradeoffs.. . .

In upcoming columns, I’ll explore those tradeoffs by writing a simple protected mode task switcher: FFTS, the Firmware Furnace Task Switcher. Although it has many of the attributes of an operating system, it’s tuned for tinkering rather than general programming. We’ll explore memory protection and paging, the CPU’s debugging hardware, interrupt and task switching response times, and the grubby details you don’t find anywhere else.

One feature should pique your interest: FFTS will use the CPU’s *Virtual 8086* mode to multitask real-mode embedded programs. We can run the same programs we’ve been using all along to get both protected-mode system functions and easy-to-code real-mode user routines. V86 programs still have most of the disadvantages of real mode, but at least a crash won’t wipe out the whole system.



| | | |
|---|---|---|
| Segment Base | 32-bit starting address | |
| Segment Limit | 20-bit size - 1 | |
| Segment Attributes | | |
| G | Granularity | 0 = limit in bytes (1 MB)<br>1 = limit in 4K chunks (4 GB) |
| D | Default op size | 0 = 16-bit<br>1 = 32-bit |
| 0 | Reserved | must be zero |
| A | Available | may be used by system firmware |
| Segment Access Rights | | |
| P | Present | 0 = not present in storage or invalid<br>1 = present and valid |
| DPL | Desc Privilege | 00 = most privileged (system)<br>11 = least privileged (user) |
| S | Seg Type | 0 = system segment<br>1 = code, data, or stack segment |
| Type | | See Figure 2 |

Figure l--Every *byte of storage accessible to the* CPU in *protected* mode must be addressed by a segment *descriptor. The* various fields specify the segment’s *attributes so the CPU can* verify each access; an improper *access causes a* protection exception... which is what protected mode is *all* about.

As in previous columns, I’ll proceed step by step with an eye on the fundamentals. There will be lots of debugging support to help isolate problems and the overall design will be simple enough that we don’t get lost in the forest. Although the Firmware Development Board’s hardware will come in handy, it’s not absolutely essential; you’ll be able to tweak the code to run without it if you like.

Because this is a firmware column, you’ll continue to see scope traces, logic analyzer records, and timing diagrams. I’ll avoid discussing many operating system design issues; if you need that sort of stuff, there are references available. Here you’ll see what happens at the point where firmware meets hardware-where microseconds matter!

The folks in the local Robotics Club are interested in FFTS because they see it as a way to get a bunch of virtual CPUs running separate real-mode robot control tasks. With any luck, we can handle a bunch of sensors and actuators with one cheap PC system board and a little interface hardware.

And, of course, if you’re building an embedded system (perhaps with a new special-purpose '386EX?) you can surely put FFTS to good use. Think of it as the kernel for your own application rather than an operating system and you’ll be on the right track.

Although C++ is the current most-favored language, FFTS will use Borland’s TASM. Assembler is a tried-and-true part of this series and, as it turns out, most of the code we’ll need is better suited to assembly language anyway. I am looking at using 32-bit C compilers for some of the more complex code later on, but we have a long way to go before that becomes critical.

The real-mode setup code will use the DOS versions of Borland C++ and TASM. As before, Paradigm’s Locate will convert the whole affair to a binary file compatible with the custom disk boot loader.

## A WHOLE NEW CPU

Each of the books in the “Reading Assignment” section at the end of this column devotes several chapters (in

a) S=O defines a System Segment. When S=O the Type field is a binary number identifying the segment. These values are the most useful ones for 32-bit programs:

| Seg Type | Meaning |
|---|---|
| 2 | Local Descriptor Table |
| 5 | Task Gate |
| 6 | 16-bit interrupt Gate |
| 9 | 32-bit Task State Segment, not busy |
| B | 32-bit Task State Segment, busy |
| C | 32-bit Call Gate |
| E | 32-bit Interrupt Gate |
| F | 32-bit Trap Gate |

b) S=l defines a code, data, or stack segment. When S=l the Type field is bitmapped with these meanings:

Bit 1 1=0 defines a data or stack segment:

| Bit | Name | Meaning |
|---|---|---|
| 11 | Executable | 0 = data segment |
| 10 | Expansion Direction | O=expand up |
| | | 1 = expand down |
| 9 | Writable | 0 = Read-only |
| | | 1 = Read/Write |
| 8 | Accessed | 0 = descriptor not used or tested |
| | | 1 = set when used by CPU |

Bit 1 1=1 defines a code segment:

| Bit | Name | Meaning |
|---|---|---|
| 11 | Executable | 1 = data segment |
| 10 | Conforming | 0 = normal privilege applies |
| | | 1 = execute at caller's privilege |
| 9 | Readable | 0 = Execute-only |
| | | 1 = Execute/Read |
| 8 | Accessed | 0 = descriptor not used or tested |
| | | 1 = set when used by CPU |

Figure 2—*The S bit and the Type field define a bewildering variety of segments. The more common values are shown here; consult fhe CPU documentation for further details.*

some cases, the entire book!) to explaining how '386 protected mode works. For now I'm going to point out just the brightest highlights before diving into the code. In future columns, I'll get into more details, but you'd be well advised to pick up a few books that have the tables and charts.

The key to understanding protected-mode programming is what's called a *segment descriptor.* Every memory address, every FAR J M P and CA L L, every interrupt handler, every task is associated with a descriptor. If it's not in a descriptor, you not only can't use it, you can't even see it.

A descriptor is simply the 8-byte memory structure shown in Figure 1. The S bit and Segment Type field specify the descriptor type and control the meaning of the other fields. Although you'd like to have contiguous address and limit fields, the barnacles prevent it: '386 descriptors are backward compatible with the 16-bit descriptors found in the '286, so the extra bits just didn't fit very neatly. Everybody gets to unscramble a three-part address field and a two-part limit field just to maintain compatibility with an obsolete CPU.

In fact, the LSL (Load Segment Limit) instruction "loads a register with an unscrambled segment limit" so you don't have to write code to figure it out. CISC to the rescue!

Descriptors fall into two broad classes: system and segment. When the S bit is 0, denoting a system segment, the descriptor contains information about tasks, interrupt handlers, call gates, and suchlike. When the S bit is 1, the descriptor maps code, data, and stack memory segments. Figure 2 shows the most useful values of the Segment Type field in each case.

The CPU uses three tables of descriptors, each selected by a separate CPU register: the Global Descriptor Table, the Local Descriptor Table, and the Interrupt Descriptor Table. While you can build descriptors outside these three tables, the CPU can't make use of them. It bears repeating: if the thing you want to use isn't covered by a descriptor in the appropriate table, the CPU cannot access it.

The GDT holds system and memory descriptors that are available throughout the system. The first entry is particularly important: it must be filled with zeros because it represents the null pointer. Any attempt to reference storage with that descriptor causes a protection exception.

When we start multitasking you'll see that each task can also have a separate LDT defining its memory and routines. LDT descriptors are available only to a single task, but each task can also access memory and routines shared with other tasks through the GDT. For the moment, though, we won't use LDT descriptors.

The IDT corresponds to the familiar real-mode interrupt vectors starting at address OOOO:OOOO. In protected mode, the "vectors" are 8-byte system descriptors that transfer control to the appropriate interrupt handler. If an IDT descriptor is invalid, the CPU will invoke an exception handler rather than simply leap into hyperspace. Of course, it's your responsibility to make the descriptors correct; if you put a bad address in the descriptor, you still get what you deserve.

In real mode, the segment registers hold the upper 16 bits of the 20-bit physical address. In protected mode, they hold a number, called a *selector,*

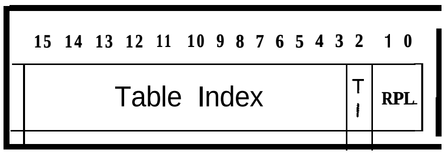


Figure 3—*In protected mode, the segmenf registers contain "selectors" rather than address bits. The Jab/e Indicator bit selects either the GDJ or IDT and fhe index field selects one of the 8K descriptor entries (hence the name). The CPU will cause a protection exception if the descriptor is invalid. The RPL bits indicate the Requestor Privilege Level, but we won't need that complexity for awhile.*

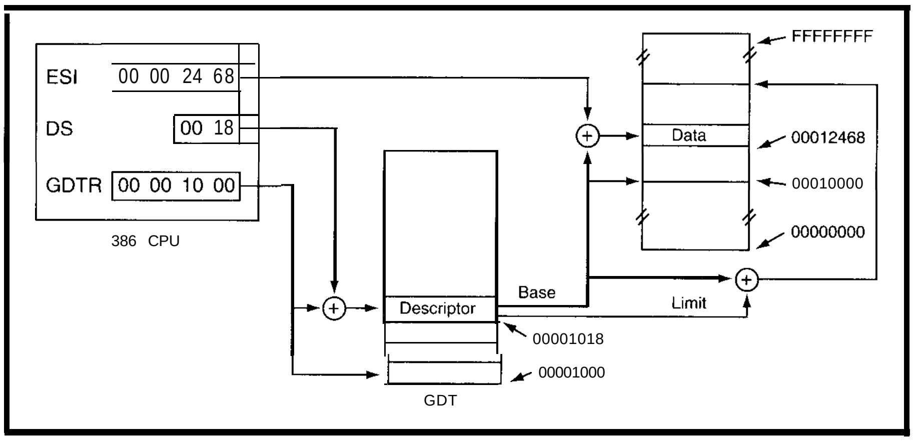


Figure 4—*Protected mode memory addressing uses far more CPU hardware than real mode. In this example, the CPU's Global Descriptor Table Register holds fhe GDT's starting address. The selector in DS identifies one of the GDT descriptors, which contains the starting address and size of the data segment. The offset in ESI (which must be less than the segment size) is added fo the starting address to create the actual memory address. The CPU holds the descriptor information in an infernal register to improve performance; otherwise each memory access would require several GDT accesses.*

that identifies a descriptor in either the GDT or LDT. Figure 3 shows the selector layout: the high-order 12 bits are an index into the GDT or IDT (which are thus limited to 8192 entries or 64K bytes maximum). If bit 2 is 0, the selector refers to the GDT; if it's 1, the CPU uses the current LDT.

The remaining two bits contain the requester privilege level, which I'll get into much later on. For now, suffice it to say that we'll give ourselves "most privileged" status and try to stay out of trouble.

Figure 4 shows how this works for the instruction MOV EAX , [ESI], which implicitly refers to segment register DS. In this example, DS holds 0x0018, which corresponds to entry 3 in the GDT. The GDT Register points to the start of the GDT at 0x00001000. Adding that value to DS produces the descriptor's memory address: 0x00001018.

The CPU fetched that descriptor from the GDT when a previous instruction loaded DS. All eight bytes of the descriptor remain in an internal CPU cache register so successive instructions don't reread the GDT. If you change the GDT, you must also reload the selectors to update the corresponding descriptor cache entries.

The CPU compares the offset in ESI, which we'll assume is 0x00002468, against the descriptor's limit field, which is 0x1FFFF, to verify that it lies within the segment. If not, the CPU causes a general protection exception to prevent an invalid memory access; you can't read or write beyond the end of a segment anymore.

Because the descriptor limit field has only 20 bits, the largest segment seems to be 1 MB. Although that's better than 64 KB, it's not really enough. The descriptor's G bit controls the "granularity" of the segment limit: when G = 1 the limit field has units of 4096 bytes (0x1000) and the maximum segment can be 4 GB long. Conversely, the smallest G = 1 segment is 4 KB, so you should set that bit only when you need it.

If the offset lies within the segment, the CPU adds the segment base address from the descriptor—0x00010000-to the contents of ESI. The result-0x00012468-is called the ***linear memory address.***

Later on we'll activate the '386 CPU's paging mechanism, which translates linear addresses into the physical addresses that actually go out on the address bus. For now, paging is disabled and linear addresses are identical to physical addresses. For '386SX CPUs, you must ensure that all physical addresses fall in the first 16 MB of memory because the chip has only 24 address lines.

Finally, the CPU fetches the contents of the 32-bit word at 0x00012468 from memory and loads it into EAX. As far as the program is concerned, it's a single instruction just like in real mode.

***Whew!***

Believe it or not, that whole process takes the same four clock cycles it does in real mode. The '386 includes quite a bit of extra hardware to compute protected mode addresses quickly. The '486 brings even more hardware to bear and fetches the value in a single cycle.

Now you can see what's so protected about protected mode. Among other things, the DS selector must be valid, the descriptor must define a readable data segment, and the offset must lie within the segment. The CPU will cause a protection exception if any of the myriad checks detects an inconsistency.

For example, you cannot load a nonreadable code segment descriptor into DS, let alone overwrite the code. You cannot load a data segment descriptor into CS, so you can't execute data. If you fill the unused GDT entries with invalid descriptors, the simple act of loading a wild pointer causes a protection exception. If that pointer happens to refer to a valid descriptor, you'll get a protection exception when you try to read or write beyond the segment limit. Yes, you can still clobber yourself, but you have to work harder at it.

There's more to protected mode than that, but, as Lao Tzu put it, "A journey of a thousand miles must begin with the first step." Here we go....

## WATCH THAT FIRST STEP!

The 80386 CPU emerges from hardware reset in real mode. Although the barnacles dictate that it must look like a fast 8086 (so it can run all that obsolete code!), the fundamental justification for real mode is that you can't enter protected mode until you prepare [at least) the GDT and IDT. Until those tables are valid, switching to protected mode will cause an immediate error.

In fact, because the error occurs in protected mode, the CPU will attempt to vector to the appropriate error handler through the IDT. The IDT is invalid, so the CPU will attempt to vector to the double-fault handler, which is also in the IDT and equally invalid. The CPU, operating on the

Listing 1—*The GDT must* be created by a program running in real *mode, so we can use ordinary* embedded C code. *This* routine *allocates a* block *of memory* on the near heap and fills in the descriptors *required by the B/OS function. The des c_norm structure maps the fields shown in Figure 1. The MakeLinear() function converts* a real-mode *far pointer info a linear 32-bit address, while Set DescAddr() fills the various address and limit fields. The segread()* function *puts fhe real mode segment register values* into the *SREGS structure so the code* can translate *them into the corresponding protected mode segment base addresses.*

```
int BuildGDT(desc_norm ** ppGDT){

desc_norm * pDesc;
int Index:
struct SREGS sregs;

   *ppGDT = calloc(GDT_LENGTH,sizeof(desc_norm));

/*- fill in the default fields */
/* skip NULL and BIOS CS entries, as required by the BIOS spec */

   for (Index = 1; Index < (GDT_LENGTH-1); ++Index){
      ((*ppGDT)+Index)->Access.SegType = 1;   // code/data/stack
      ((*ppGDT)+Index)->Access.Present = 1;   // always present
      ((*ppGDT)+Index)->Access.ReadWrite = 1; // data=writable,
                                              //      code=readable
   }

/*- fill in the table entries                 */

   segread(&sregs);    // get current seg regs
   pDesc == *ppGDT;    // aim at the GDT starting address
```

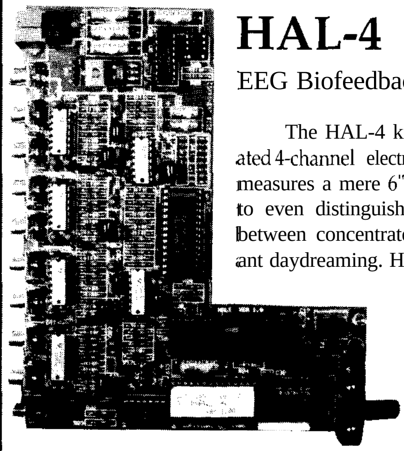
(continued)

"three faults and you're out" principle, enters shutdown mode. The PC system board responds to CPU shutdown with a hardware reset.. .which starts the cycle over again.

The '386 manuals cover the intricate dance needed to activate protected mode when your code is running on a bare system. In our case, however, there's an easier way: BIOS Int 15, AH=89 is the Official "Switch to Protected Mode" function. You prepare a GDT with a few specific descriptors, aim the IDT at your exception handlers, invoke the BIOS function, and it returns control to you with the CPU in protected mode.

The motivation for using the BIOS function is the same as always: manufacturers (presumably) invest a great deal of effort encrusting their system BIOS code with compatibility barnacles. No matter what oddness lies at the hardware level, you can be reasonably sure the BIOS will set things up so the CPU works correctly after the mode switch.

**CIRCUIT CELLAR KITS**



**HAL-4**

EEG Biofeedback Brainwave Analyzer

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

HAL's operation is 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 frequency, amplitude, and phase components, the results are graphically displayed in real time for each side of the brain.

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

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

Sonar Ranging Experimenter's Kit

Targeting ◆ Ranging ◆ Machine Vision

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

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

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

**CHECK OUT THE NEW CIRCUIT CELLAR HOME CONTROL SYSTEM**

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

**Call and ask about the *HCS II***

To order the products shown or to receive a catalog,

**call: (203) 875-2751 or fax: (203) 872-2204**

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

#123

The barnacles on Int 15, AH=89, date back to the Original IBM AT, so the GDT must be compatible with an 80286 CPU rather than the 80386 we're using. This is no problem, but it would help if the references mentioned this. They don't. Trust me. I've looked.

Listing 1 shows the code that creates the GDT structure. I've removed the `printf()` statements to save some space, but the complete code (available on the BBS) produces the display shown in Figure 5.

Compare the GDT values shown in Figure 5 with the descriptor structure shown in Figure 1. The descriptors are listed in 16-bit format with the low-order word first. The CS/DS/ES descriptor limits are all FFFF and the base addresses match up with the values created by the Locate utility.

The GDT and IDT alias entries allow access to the memory holding those tables as ordinary data. Remember that you cannot access memory without a valid data segment descriptor: if you want to change the GDT or IDT, you must create a read/write data segment descriptor for that chunk of memory.

I won't describe the IDT structure in this column. Basically, I created a single handler to catch all 256 possible interrupts and show the interrupt ID on the FDB's LED display. The BIOS disables all interrupts before entering protected mode, so the handler will not get control if everything works.

Listing 1—*continued*

```
++pDesc;            // step to the GDT alias
SetDescAddr(pDesc,
    MakeLinear(*ppGDT),
    GDT_LENGTH * sizeof(desc_norm));

++pDesc;            // step to the IDT alias
SetDescAddr(pDesc,
    MakeLinear(pIDT),
    IDT_LENGTH * sizeof(desc_gate));

++pDesc;            // step to the DS descriptor
SetDescAddr(pDesc,
    ((LADDR)sregs.ds << 4) + (LADDR)(0L),
    (DWORD)(0x10000L));

++pDesc;            // step to the ES descriptor
SetDescAddr(pDesc,
    ((LADDR)sregs.es << 4) + (LADDR)(0L),
    (DWORD)(0x10000L));

++pDesc;            // step to the SS descriptor
SetDescAddr(pDesc,
    ((LADDR)sregs.ss << 4) + (LADDR)(0L),
    (DWORD)(0x10000L));

++pDesc;            // step to the CS descriptor
SetDescAddr(pDesc,
    ((LADDR)sregs.cs << 4) + (LADDR)(0L),
    (DWORD)(0x10000L));

pDesc->Access.Executable = 1        // code is executable
return 0;
```

Once the GDT and IDT are set up, the code shown in Listing 2 handles the transition to protected mode. This function will not return to the caller because Borland C just doesn't expect to run in protected mode. We'll have to get those V86 tasks running before real-mode C works again.

Although the BIOS spec says the function will return with the carry flag set on error, my experience has been that any failures occur on the protected mode side of the border and result in protection exceptions. If you find the system rebooting, it's an indication that it entered protected mode with an invalid IDT that prevents the error handler from gaining control.

Blinking an LED may not sound like much compared to running a protected-mode operating system. After you've checked out some of the references and looked through the BBS code, though, I think you'll feel pretty good about watching it blink, because there's a lot of information behind it.. .give it a shot!

```
Journey to the Protected Land... Step 1

Allocating IDT at 2000:040C
 Re-vector table  1000:04C6 = 000104C6
 First IDT entry  04C6 0030 8600 0000
 Second           04CE 0030 8600 0000
Allocating GDT at 2000:0C10
 GDT NULL         0000 0000 0000 0000
 GDT alias        003F 0C10 9202 0000
 IDT alias        07FF 040C 9202 0000
 DS               FFFF 0000 9202 0000
 ES               FFFF 0000 9202 0000
 SS               FFFF 0000 9202 0000
 CS               FFFF 0000 9A01 0000
 BIOS CS          0000 0000 0000 0000
```

Figure 5—*The BIOS "Switch to Protected Mode" function requires a GDT loaded with 16-bit segment descriptors mapping the same addresses as the real mode segments and an IDT loaded with inferrupf handler descriptors. This screen dump shows the settings produced by the sample code this month. Compare these fields with the descriptor format shown in Figure 1. The IDT and revector fable will be covered in upcoming columns.*

## STEPS, BUGS, AND ERRATA

One advantage of a standard PC system board is that you're reasonably certain the hardware works correctly. It's not like an R&D project where you are part of a team inventing firmware and hardware in parallel; locating bugs and deciding who gets to fix them may require weeks of hellish effort.

Real-mode programming is about as stable as you can get because the PC compatibility barnacles force manufacturers to test their hardware against the universe of DOS applications. With precious few exceptions, you can do real-mode programming "by the book" and get exactly the results you expect.

Listing 2—*This code hand/es the transition to protected mode and displays status outputs so you can see what happened. It must wait for the floppy drive motors to stop because fhe timer interrupt handler isn't valid in protected mode. The protected mode code blinks an LED and loads fhe FDB's DIP switches info ES; if the bits are not a valid* GDJ *selector, the* CPU *will* ***cause*** *a general protection failure* and *invoke the Int 0D handler through the IDT. Make sure the DIP switches are all OFF before running this code.*

```
        PROC    PMEntry
        ARG     VectorA:WORD,VectorB:WORD
        PUBLIC  PMEntry

        MOV     AL,001h            ; flag entry here
        MOV     DX,SYNC_ADDR
        OUT     DX,AL

        MOV     AX,NOT 0101h       ; preset the digits to dashes
        MOV     DX,LED_ADDR
        OUT     DX,AX

;- wait for diskette motor to stop spinning

        MOV     AX,0040h           ; aim at motor status
        MOV     ES,AX
        MOV     SI ,003Fh          ; at 0040:003F
@Motors:
        MOV     AL,[ES:SI]
        TEST    AL,03h             ; drives A & B
        JNZ     @@Motors

;- set up the registers

        MOV     AX,DS              ; set ES:SI to GDT base address
```

(continued)

In contrast, protected mode programming is a minefield. Consider this paragraph from the "Incompatibilities and Bugs" chapter of Hummel's ***The Processor and Coprocessor:***

"...On some versions of the 80386, if the operand of the VERR instruction is not accessible and none of the instructions following the VERR in the prefetch queue is a JMP, CALL, or has a memory operand, then the processor will hang after executing the VERR. The processor remains stopped until an INTR, NMI, or RESET occurs. The system timer interrupt will normally unhang the system...."

Makes you sick just to think about it, hmm?

Many PM CPU bugs occur only in peculiar, difficult-to-reproduce situations. If you are writing for a particular CPU, you can tailor your code around hardware problems if you know what they are. Slater's ***Microprocessor Report*** is famous for printing bug lists, but at $500/year, you need real justification to put it on your reading list.

**The *Soft-Scope*® /*CSiMON* debugger and *CSi-Locate*™ combined with your existing compiler create a complete embedded systems development environment**

***Absolute or Hex files***

New ***CSi-Locate*** builds absolutely located files or hex files from applications compiled using Microsoft, Borland, or Watcom tools. Soft-Scope also supports Intel, Metaware, and Symantec tools.

***Easy Configuration***

Configure a ***CSiMON*** ROM monitor in minutes for most targets.

***Versa tile***

For the entire 8086 ***family*** of processors, in real or protected mode.

***Cost effective***

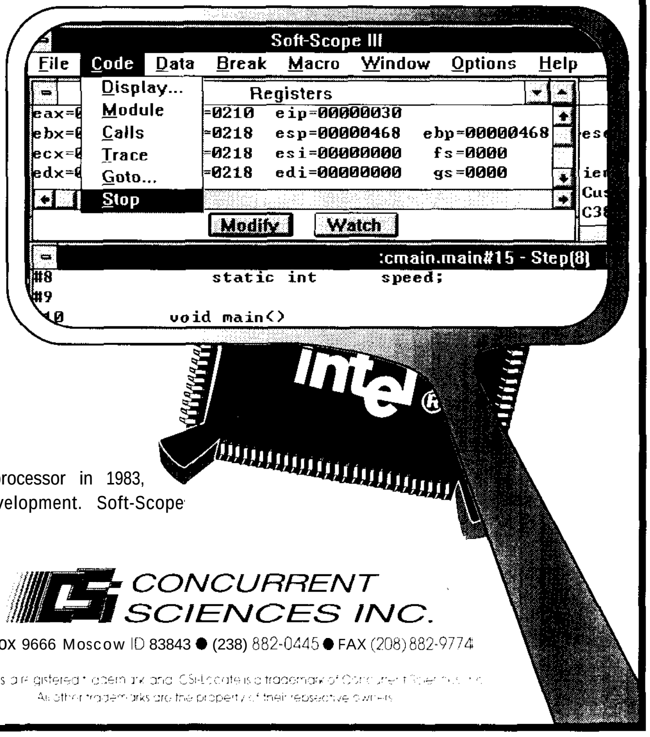
Include the *royalty-free* CSiMON ROM monitor as part of your product.

***Real-time support***

Intel's iRMX, Industrial Programming Inc's. MTOS-UX and JMI Software Systems, C Executive fully support ***Soft-Scope***.

***Established***

Concurrent Sciences developed the original source-levei debugger for the 8086 processor in 1983, and is known is known industry-wide as the expert in 32-bit protected-mode development. Soft-Scope fully supports the Intel386 EX processor.



```
eax=0   =0210  eip=00000030
ebx=0   =0218  esp=00000468  ebp=00000468   es
ecx=0   =0218  esi=00000000   fs=0000
edx=0   =0218  edi=00000000   gs=0000       ie
                                            Cu
                                            C3
#8              static int      speed;
#9
10          void main()
```

"We have about 750,000 lines of source code in our product, running on an 80486 in both 16- and 32-bit protected mode, and we are very pleased with how well Soft-Scope deals with multitasking and mixed 16/32-bit code."

Gerd Hoeren Integrated Measurement Systems. Beaverton OR

CONCURRENT SCIENCES INC.

PO Box 9666 Moscow ID 83843 ● (238) 882-0445 ● FAX (208) 882-9774

Soft-Scope is a registered trademark and CSi-Locate is a trademark of Concurrent Sciences, Inc.
All other trademarks are the property of their respective owners.

#124

Just to make things worse, the bug lists I've seen are for Intel CPUs. Most '386SX PC system boards, including the one I'm using for this series, sport AMD CPUs. I tried to get errata lists from AMD's Tech Support line to no avail; unless you're a Big Customer, there's not much information available.

As with all manufactured items, CPUs are continually changed to improve manufacturing yield, fix bugs (really!), and take advantage of new processes. Each chip revision, called a "step" (at least by Intel), has its own collection of glitches. The '386 and later CPUs load their chip ID and revision number into DX as part of the hardware reset. There's a new Pentium instruction to get those values without having to jump through hoops, but that doesn't help us here.

(Hint: the Worst Hack leaves EDX unchanged. If you really want those numbers you can get them. Just don't blame me, OK?)

Even if I could identify the company, CPU, and revision level, I can't code workarounds for all possible problems and be sure they actually work. Not only do I lack a slave labor pool, I don't have every PC clone ever made on the shelf ready for testing.

Solving the general case seems impossible. I'm going to write PM code "by the book" using the head-in-the-sand assumption that we're all using relatively new CPUs free from the nasty old bugs. Please send me a note if you know of any nasty new bugs.

FFTS will include debugging support to identify problems as they crop up, which is surely more valuable than an easily outdated bug list. As you should know by now, nearly all bugs are in the software anyway!

Listing 2-continued

```
        MOV     ES,AX
        MOV     SI,[pGDT]
        MOV     BH,[BYTE PTR VectorA]; IRQ 0:7
        MOV     BL,[BYTE PTR VectorB]; IRQ 8:15

        MOV     AL,003h         ; mark start of BIOS function
        OUT     DX,AL

;- real mode

        MOV     AH,89h          ; BIOS function
        INT     15h             ; make the transition

;- protected mode?

        MOV     AL,080h         ; mark just after the function
        MOV     DX,SYNC_ADDR
        OUT     DX,AL

;- decide what mode we're in

        JNC     @PM

;- still in real mode, so flag the failure

        XOR     CX,CX
        MOV     DX,SYNC_ADDR
@@Fail:
        IN      AL,DX           ; by blinking the whole port
        NOT     AL
        OUT     DX,AL

        MOV     BL,5
@@Fail1: LOOP   @@Fail1         ; delay for a while
        DEC     BL
        JNZ     @@Fail1
        JMP     @@Fail

;- protected mode!

@@PM:
        XOR     CX,CX           ; initialize the delay counter
        MOV     DX,LED_ADDR     ; show a couple of Ps
        MOV     AX,NOT 6767h
        OUT     DX,AX
@@Pass:
        MOV     DX,SYNC_ADDR    ; blink bit 6 on the parallel port
        IN      AL,DX
        XOR     AL,40h
        OUT     DX,AL
        MOV     BL,5

@@Punt: LOOP    @Punt           ; delay for a while
        DEC     BL
        JNZ     @@Punt
        MOV     DX,SW_ADDR      ; check switches to blow up
        IN      AX,DX
        NOT     AX              ; . . . ON = 1
        MOV     ES,AX           ; GP fault = 13 decimal = 0D hex
                                ; if AX is not valid selector
        JMP     @@Pass
        ENDP    PMEntry
```

## READING ASSIGNMENT

We are now leaving the applications programming world behind. Most of the popular bookstore references dismiss the issues we'll encounter by noting that only "systems programmers" must worry about them. Well, OK, if you don't want to know how the machinery works, don't pay any attention to this stuff.. .

Intel, of course, has the definitive 80x86 CPU documentation and their Literature Hotline (800/548-4725, P.O. Box 7641, Mt. Prospect, IL 60056-7641) is the place to get it. They'll send you a catalog *(Customer Literature Guide,* Order Number 210620-030) free, but the rest of their books cost real money plus 15% shipping.

80386 *System Software Writer's Guide,* Order Number 23 1499-00 **1, $18.00.** This covers the grubby low-level details required to bolt a protected-mode operating system onto a '386 CPU. It has a wealth of "Oh, so *that's* what they meant!" info-nuggets. Not to be missed.

*386DX Microprocessor Programmer's Reference Manual,* Order Number 230985003, $25.95. This covers both applications and systems programming, but with a different slant than the *System Software Writer's Guide.* Get both of 'em. You can also get the 386SX manual, but it's essentially the same as the 386DX for our purposes.

*Microprocessors: Volume 1,* Order Number 230843-011, $24.95. More than you really want to know about 80386, 80286, and 8086 hardware. There are also *Hardware Reference Manuals* for each CPU if you need even more detail.

Robert Hummel's *The Processor and Coprocessor* remains a ready desk reference. It covers much of the same ground as the Intel *Programmer's Reference,* but the approach is quite different. Intel's doc, for example, says Word Zero about bugs. Ziff-Davis Press, ISBN 1-56276-016-5, $49.95.

Although the next two books have nothing to do with protected-mode programming, they are valuable nonetheless. If you write any code at all, for any reason, read them; they're that good.

Steve Maguire's *Writing Solid Code* is a well-written introduction to *Microsoft's Techniques for Developing Bug-Free C Programs.* I was pleased to find I already use some of the techniques, but he pointed out several flaws in my thinking. Your code will benefit, too. Microsoft Press (of course), ISBN 1-55615-551-4, $24.95.

Steve McConnell's *Code Complete* is a much weightier tome on "A Practical Handbook of Software Construction." The chapters include modular design, data naming conventions, control flow organization, construction tools, quality improvement, and code tuning. If it weren't so well-written, it'd be overwhelming. Buy it and use it liberally. Microsoft Press, ISBN 1-55615-484-4, $35.00.

## RELEASE NOTES

The BBS code will flip your '386SX into 16-bit protected mode while toggling bits on the printer port and FDB LED display. This is a simple compatibility test: if it doesn't work, there is something badly wrong somewhere. It may work with '486 systems, but I haven't tried it.

I used Borland C and TASM with Paradigm's Locate to create the binary file. You'll also need the boot sector loader we used in Issue 41.

Credit to Jim White for suggesting the catchy "Journey to the Protected Land" series title. Peter Holtzleitner, who regularly dials into the BBS from Austria, sent a complete V86 monitor program written in German with translations of many of the comments. Derry Bryson contributed another V86 monitor (in English!). And thanks to all of the BBS regulars for helping me figure out how to cover this elephantine topic in a reasonable manner. Any errors, of course, are of my own invention.

Next month: 32 bits or bust! ❑

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

## SOFTWARE

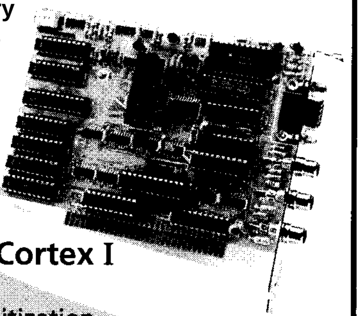
Software for this article is available from the Circuit Cellar BBS and on Software On Disk for this issue. Please see the end of "ConnecTime" in this issue for downloading and ordering information.

## I R S

413 Very Useful
414 Moderately Useful
415 Not Useful

# Video Frame Grabber

- $495 Including Software with "C" Library
- Half Slot Card for Compact Applications
- Real Time Imaging with Display Output
- 8 Bit (256 Gray Levels)



***FEATURES:***
- Single 512 X 484 Image or Four 256 X 242 Images
- External Trigger
- Input Look Up Table
- Low Power Option Available
- Elegant Software Interface
- <10 nsec Pixel Jitter Means Accurate Digitization
- EISA (PC) Bus and STD Bus Products Available
- RS-170 and CCIR Video Formats Available
- Binary and TIFF File Formats

**The Cortex I is used in machine vision, industrial control, medical, security and scientific applications around the world.**

**ImageNation strives to delight customers with quality products and personal service at a competitive price.**

***Call today for volume pricing or to discuss your application.***

*ImageNation Corporation*
*Providing Imaging Solutions*

P.O. Box 276
Beaverton, OR 97075
(503) 641-7408

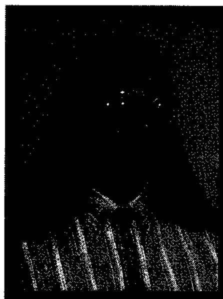
**FAX (503) 643-2458 • (800) 366-9131**

#125

Jeff Bachiochi

# Virtual Reality Requires Real Data

## Part II-A Simple Matter of Software



Got those exo-skeletons built yet? If so, now is the time to do some experimenting and data collection. It wouldn't be a complete project if the software wasn't covered. So Jeff steps back into the virtual world with a view from software.

Viewing 3-D objects in a two-dimensional world is like trying to be judge, ump, or referee while watching sports on TV. What you see does not always represent what is really happening. You have a single point of reference on the scene. While instant replay can give us a different view on the scene, it too is limited to wherever the camera happens to be located in relationship to the play.

In order to visually verify 3-D movements (movements which cross your field of vision both vertically and horizontally, as well as move toward and away from you), these movements must be seen from different perspectives. The most widely used perspectives are top, front, and side views. However, each of these views gives only two sides of a three-sided story, but the views are accurate because all dimensions are true and not distorted in an attempt to show depth (the third dimension).

Most humans can see in 3-D perspective. Our eyes, although only inches apart, obtain two different views. Subconsciously, through the use of triangulation, our brain categorizes each object we see by depth. Depth is determined by the difference in the object's angles from each eye. When looking at an image on a flat screen, both eyes see the same image and the only depth detected is how far away the screen is, even when position is shifted. By giving the image a bit of (vanishing point) perspective, a feeling of depth can be imitated.

In an effort to keep things simple, I wanted to produce the three basic views of my "Virtual Arm." These

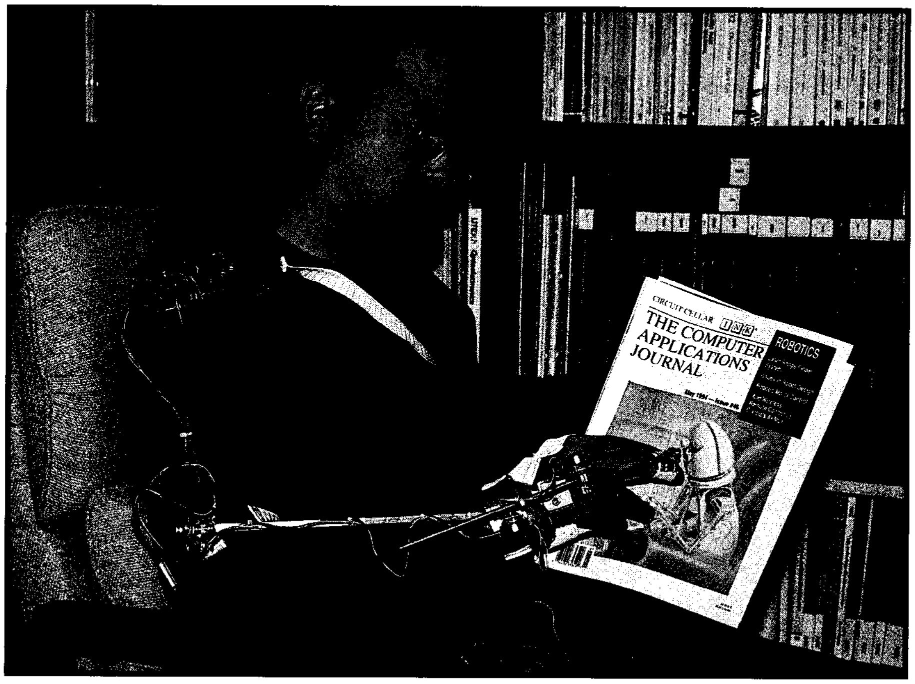


Photo I--Now *Jeff's* computer can keep *track of his every movement while he checks out his favorite magazine.*

**Listing 1--The exoskeleton support** *software not only collects data but* displays if on *the screen. It even allows you to try it out without the actual hardware.*

```
10 PRINT "Are you using an exoskeleton as described in FTB47?"
20 I$ = INKEY$: IF (I$ = "") THEN GOTO 20
30 IF (I$ = "Y") OR (I$ = "y") THEN REAL = 1 ELSE REAL = 0
40 IF (REAL = 1) THEN OPEN "COM1:9600,N,8,1,CD0,CS0,DS0,OP0,RS"
    FOR RANDOM AS #1
50 CLS : SCREEN 2

60 DIM  PX(11), PY(11), PZ(11), OPX(11), OPY(11), OPZ(11)
70 DIM  FP(11), TP(11), YAWT(11), ROLLT(11), PITCHT(11)
80 DIM  YAWO(11), ROLLO(11), PITCHO(11)
85 DIM  OYAWT(11), OROLLT(11), OPITCHT(11)
90 cx = -30: CY = 3: CL = 0: cc = 0

100 TVPX = 159: TVPZ = 50: RVPZ = 479: RVPY = 199: FVPX = 159:
     FVPY = 199
110 AR = 2: YAWT(0) = 0: ROLLT(0) = 90: PITCHT(0) = 0
120 FP(1) = 0: TP(1) = 1: YAWO(1) = 0: ROLLO(1) = 0:
     PITCHO(1) = 0: L(1) = 40
130 FP(2) = 1: TP(2) = 2: YAWO(2) = 0: ROLLO(2) = 180:
     PITCHO(2) = 0: L(2) = 25
140 FP(3) = 2: TP(3) = 3: YAWO(3) = 0: ROLLO(3) = 0:
     PITCHO(3) = 0: L(3) = 20
150 FP(4) = 3  TP(4) = 4: YAWO(4) = 0: ROLLO(4) = 0:
     PITCHO(4  = 0: L(4) = 5
160 FP(5) = 4  TP(5) = 5: YAWO(5) = 0: ROLLO(5) = 0:
     PITCHO(5  = 0: L(5) = 5
170 FP(6) = 5  TP(6) = 6: YAWO(6) = 0: ROLLO(6) = 0:
     PITCHO(6  = 0: L(6) = 5
180 FP(7) = 4  TP(7) = 7: YAWO(7) = -90: ROLLO(7) = 0:
     PITCHO(7  = 0: L(7) = 5
190 FP(8) = 5: TP(8) = 8: YAWO(8) = 0: ROLLO(8) = 40:
     PITCHO(8) = 0: L(8) = 5
200 FP(9) = 5: TP(9) = 9: YAWO(9) = 0: ROLLO(9) = 20:
     PITCHO(9) = 0: L(9) = 5
210 FP(10) = 5: TP(10) = 10: YAWO(10) = 0: ROLLO(10) = 340
     PITCHO(10) = 0: L(10) = 5
220 CONST Q = 3.141593 / 180

230 LOCATE 2, 2: PRINT "Top View (YAW Y-Axis)";
240 LOCATE 14, 2: PRINT "Front View (ROLL X-Axis)";
250 LOCATE 14, 42: PRINT "Right View (PITCH Z-Axis)"
260 LINE (0,0)-(319,0): LINE (639, 99)-(639, 199):
     LINE (639, 199)-(0,199)
270 LINE (0,199)-(0,0): LINE (0,99)-(639, 99):
     LINE (319, 0)-(319,199)

1000 LOCATE 1, 42: PRINT "SERIAL DATA"
1010 IF REAL = 0 THEN GOTO 1030
1020  INPUT #1, V$(0), V$(1), V$(2), V$(3), V$(4), V$(5)
      V$(6), V$(7)

1030 LOCATE 1, 42: PRINT "UPDATING JOINT ARRAYS": FOR Z   1 TO 10
1040 PITCHA(Z) = 0: ROLLA(Z) = 0: YAWA(Z) = 0
1050 NEXT Z

1060 PITCHA(2) = VAL(V$(7))
1070 ROLLA(2) = VAL(V$(6))
1080 YAWA(2) = VAL(V$(5))
1090 YAWA(3) = VAL(V$(4)) * -1
1100 PITCHA(4) = VAL(V$(3)) * -1
1110 ROLLA(4) = VAL(V$(2))
1120 YAWA(4) = VAL(V$(1))
1130 YAWA(6) = VAL(V$(0))
1140 YAWA(8) = YAWA(6): YAWA(9) = YAWA(6) : YAWA(10) = YAWA(6)

1150 FOR Z = 1 TO 10
```

*(continued)*

views will show movement of the arm from all directions at once. To synchronize the "real data" from the exoskeleton (discussed in Part 1 last month) with this graphic virtual representation, the data measurements must start from some known point. The "zeroing mode" yields a bit of flexibility in determining the "at rest" position. As long as both the input device and the PC program (see Listing 1) agree, all should be happy. I chose to use the arm hanging down at my side with the palm back and grip open as the "zero" or "calibration" point. This will be the initialized position when the program is started.

## THE EXOSKELETON'S DATA

It is essential to understand how the data output by the exoskeleton reflects real movements so we can reconstruct the "Virtual Arm." Refer to Figure 1 and Photo 1 to identify the position of each point (or joint) and angle measuring sensor (potentiometer).

Notice that the Z-axis is used as the continuous and common axis throughout the length of the body. This simplifies the interaction between axis movements. The Virtual Arm is constructed from lines drawn between two points in stick figure fashion. The points' positions are determined first by the program's initial "zero" position and thereafter by data received from the exoskeleton.

## DEFINING THE VIRTUAL WORLD

To model the arm, I chose to start with a connection from the ground [point 0) to the shoulder (point 1). This simulates the body (lever 1) from the feet to the shoulders. The connection from the shoulder (point 1) to elbow (point 2) becomes lever 2 and simulates the upper arm. The lower arm is from point 2 to point 3, and the wrist is lever 3. The hand is made up of four main levers. The first (lever 4) is from the wrist to the thumb joint (point 4), and the second (lever 5) is from the thumb joint to the finger joint (point 5). Lever 6 represents the fingers from the joint at point 5 to the tip at point 6. The last lever (7) is made from the thumb at the joint at point 4 to the tip at point 7. See this illustrated in Figure 1. Just for aesthetics, I added three extra fingers (little, ring, and pointer). All fingers will move together.

Every joint is represented as two levers connected between three points. Each joint's position is based on the previous joint's position and has (potentially) three rotational axes. The Z-angle (pitch) is a CW (clockwise) or CCW (counterclockwise] rotation as viewed from the side. The X-angle (roll) is a CW or CCW rotation as viewed from the front. And the Y-angle (yaw) is a CW or CCW rotation as viewed from the top. In reality, the arm's movements are limited to partial rotation and in only particular axes depending on the joint. This reduces the computations from three dimensions times five joints (15) down to just eight (the number of positional sensors being used).

The first calculations determine the position of each joint in the universe and are derived using a polar coordinate system (Figure 2). This designates the first point as the polar origin or center of the universe. To find the second point's location, use the angular directions from the last point (in this case the polar origin) using the X- and Y-angles (based on the previous Z-angle-assumed to be 0 for the first calculation) and move outward in that direction by the length of the body part being simulated. The Z-angle will affect the next point's X- and Y-angles.

By placing a joint at the center of the universe, the movement of a second joint pivoting about the first affects what is viewed from the three perspectives: top, front, and side.

Viewing from the front, the point's "width" position is affected by the Y-angle and the Z-angle, and the "height" position is affected by the X-angle and the Z-angle.

FVPX( 1) [front view width X] = FVPX(0) [joint X-axis point] + L( 1) [full length of the lever] x sin(Y-angle) [or sin(Z-angle)]

FVl?Y( 1) [front view height Y] = FVPY(0) [joint Y-axis point] + L( 1) [full length of the lever] x cos(X-angle) (or cos(Z-angle)).

Listing 1-continued

```
1160 YAWA(Z) = YAWA(Z) + YAWO(Z)
1170 ROLLA(Z) = ROLLA(Z) + ROLLO(Z)
1180 PITCHA(Z) = PITCHA(Z) + PITCHO(Z)
1190 IF (D <> 4) THEN 1210
1200 LOCATE 1 + Z, 42: PRINT Z: "XO+A= "; INT(ROLLA(Z));
      "YO+A= "; INT(YAWA(Z)); "ZO+A= "; INT(PITCHA(Z))
1210 NEXT Z

1220 IF (D <> 1) THEN GOTO 2000
1230 FOR X = 0 TO 7
1240 LOCATE 2 + X, 42: PRINT "DATA": X; "="; VAL(V$(X));"    "
1250 NEXT X

2000 FOR Z = 1 TO 10
2010 OPX(FP(Z)) = PX(FP(Z)): OPX(TP(Z)) = PX(TP(Z))
2020 OPZ(FP(Z)) = PZ(FP(Z)): OPZ(TP(Z)) = PZ(TP(Z))
2030 OPY(FP(Z)) = PY(FP(Z)): OPY(TP(Z)) = PY(TP(Z))
2040 NEXT Z
2050 ocx = cx: OCY = CY: OCZ = cz

3000 LOCATE 1, 42: PRINT "CALC ANGLE SHIFTS & SCREEN COORDINATES"
3010 FOR Z = 1 TO 10
3020 I$ = INKEY$
3030 IF (I$ = "") THEN GOTO 3250
3040 IF (I$ = "D" OR I$ = "d") THEN D = 1
3050 IF (I$ = "n" OR I$ = "N") THEN D = 0: CLS
3060 IF (I$ = "A" OR I$ = "a") THEN D = 2
3070 IF (I$ = "p" OR I$ = "P") THEN D = 3
3080 IF (I$ = "0" OR I$ = "o") THEN D = 4
3090 IF (REAL = 1) THEN GOTO 3250
3100 IF (I$ = "1") THEN V$(0) = STR$(VAL(V$(0)) + 10):
      LOCATE 2, 42: PRINT "1-FINGER O/C "; VAL(V$(0))
3110 IF (I$ = "!") THEN V$(0) = STR$(VAL(V$(0)) 10):
      LOCATE 2, 42: PRINT "1-FINGER O/C "; VAL(V$(0))
3120 IF (I$ = "2") THEN V$(1) = STR$(VAL(V$(1)) + 10):
      LOCATE 3, 42: PRINT "2-WRIST O/C "; VAL(V$(1))
3130 IF (I$ = "@") THEN V$(1) = STR$(VAL(V$(1)) 10):
      LOCATE 3, 42: PRINT "2-WRIST O/C "; VAL(V$(1))
3140 IF (I$ = "3") THEN V$(2) = STR$(VAL(V$(2)) + 10):
      LOCATE 4, 42: PRINT "3-WRIST L/R "; VAL(V$(2))
3150 IF (I$ = "#") THEN V$(2) = STR$(VAL(V$(2)) 10):
      LOCATE 4, 42: PRINT "3-WRIST L/R "; VAL(V$(2))
3160 IF (I$ = "4") THEN V$(3) = STR$(VAL(V$(3)) + 10):
      LOCATE 5, 42: PRINT "4-WRIST TWIST "; VAL(V$(3))
3170 IF (I$ = "$") THEN V$(3) = STR$(VAL(V$(3)) 10):
      LOCATE 5, 42: PRINT "4-WRIST TWIST "; VAL(V$(3))
3180 IF (I$ = "5") THEN V$(4) = STR$(VAL(V$(4)) + 10):
      LOCATE 6, 42: PRINT "S-ELBOW O/C "; VAL(V$(4))
3190 IF (I$ = "%") THEN V$(4) = STR$(VAL(V$(4)) 10):
      LOCATE 6, 42: PRINT "5-ELBOW O/C "; VAL(V$(4))
3200 IF (I$ = "6") THEN V$(5) = STR$(VAL(V$(5)) + 10):
      LOCATE 7, 42: PRINT "G-SHOULDER F/B "; VAL(V$(5))
3210 IF (I$ = "^") THEN V$(5) = STR$(VAL(V$(5)) 10):
      LOCATE 7, 42: PRINT "6-SHOULDER F/B "; VAL(V$(5))
3220 IF (I$ = "7") THEN V$(6) = STR$(VAL(V$(6)) + 10):
      LOCATE 8, 42: PRINT "7-SHOULDER U/D "; VAL(V$(6))
3230 IF (I$ = "&") THEN V$(6) = STR$(VAL(V$(6)) 10):
      LOCATE 8, 42: PRINT "7-SHOULDER U/D "; VAL(V$(6))
3240 IF (I$ = "8") THEN V$(7) = STR$(VAL(V$(7)) + 10):
      LOCATE 9, 42: PRINT "8-SHOULDER TWIST "; VAL(V$(7))
3250 IF (I$ = "*") THEN V$(7) = STR$(VAL(V$(7)) 10):
      LOCATE 9, 42: PRINT "8-SHOULDER TWIST "; VAL(V$(7))

3260 YAWT(TP(Z)) = YAWT(FP(Z)) +
                   (YAWA(TP(Z)) * COS(PITCHT(FP(Z)) * Q)) +
                   (ROLLA(TP(Z)) * SIN(PITCHT(FP(Z)) * Q))
```

(continued)

See the front view in Figure 3 for a visual rendering of these ideas.

Viewing from the side, the point's "depth" position is affected by the X-angle and the Z-angle, while the "height" position is affected by the X-angle and the Z-angle.

RVPZ(1) [right view depth] = RVPZ(0) [joint Z-axis point] + L(1) [full length of the lever] x cos(X-angle) (or cos(Y-angle))

RVPY(1) [right view height] = RVPY(0) [joint Y-axis point + L(1) [full length of the lever] x cos(X-angle) [or cos(Z-angle)]

See the side view in Figure 3 for a visual representation of the points viewed from this perspective.

Viewing from the top, the X-axis point's "width" position is affected by the Y-angle and the Z-angle, and the "depth" position is affected by X-angle and the Y-angle.

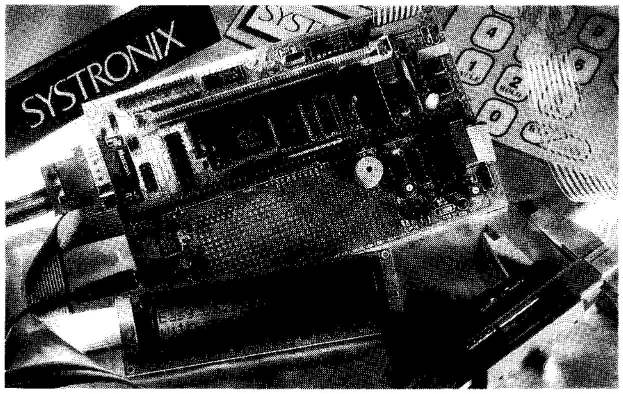
The TVPX(1) [top view width] = TVPX(0) [joint X-axis point] + L(1) [full

**Listing 1-continued**

```
3270 ROLLT(TP(Z)) = ROLLT(FP(Z)) +
                    (ROLLA(TP(Z)) * COS(PITCHT(FP(Z)) * Q))
                    (YAWA(TP(Z)) * SIN(PITCHT(FP(Z)) * 0))
3280 PITCHT(TP(Z)) = PITCHT(FP(Z)) + PITCHA(TP(Z))
3290 IF (D <> 2) THEN GOT0 3330
3300 LOCATE 1 + Z, 42: PRINT Z: "TXA=": INT(ROLLT(TP(Z)));
3310 PRINT "TYA="; INT(YAWT(TP(Z)));
3320 PRINT "TZA="; INT(PITCHT(TP(Z)))

3330 PX(TP(Z)) = PX(FP(Z))
                (COS(YAWT(Z) * Q) * COS(ROLLT(Z) * Q) * L(Z))
3340 PY(TP(Z)) = PY(FP(Z)) + (SIN(ROLLT(Z)*Q)*  L(Z))
3350 PZ(TP(Z)) = PZ(FP(Z)) + (SIN(YAWT(Z) * Q) * L(Z))
3360 IF (D <> 3) THEN GOT0 3380
3370 LOCATE 1 + Z, 42: PRINT Z; "PX="; INT(PX(TP(Z))); "PY=";
      INT(PX(FP(Z))); "PZ="; INT(PZ(TP(Z)))
3380 NEXT Z

3390 IF CC = 3 THEN CX = PX(6) + INT(.5 * (PX(7)  PX(6)))
3400 IF CC = 3 THEN CY = PY(6) + INT(.5 * (PY(7)  PY(6)))
3410 IF CC = 3 THEN CL = PZ(6) + INT(.5 * (PZ(7)  PZ(6)))
3420 IF ABS(PX(7) CX) < 5 AND ABS(PY(7) CY) < 5 AND
      ABS(PZ(7) CL) < 5 THEN CC = CC OR 1 ELSE CC = CC AND 2
3430 IF ABS(PX(6) CX) < 5 AND ABS(PY(6) CY) < 5 AND
      ABS(PZ(6) CL) < 5 THEN CC = CC OR '2 ELSE CC = CC AND 1

4000 LOCATE 1, 42: PRINT "ERASING/REDRAWING SCREEN"
4010 FOR Z = 1 TO .0
```

(continued)

NEW! UNIVERSAL DALLAS DEVELOPMENT SYSTEM from **$199!**



- It's a complete 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 300 mA 12V relay driver outputs.
- Power with 5VDC regulated or 6-13 VDC unregulated
- Large prototyping area, processor pins routed to headers
- Optional enclosures, keypads, LCDs, everything you need
- BCl51 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

#127

**TW523**
**PowerLine Interface Kits**

Interface your computer to transmit and receive X-IO codes over your AC power line. Real-time control. Each kit contains the X-10 TW523, cable, external DB25/RJ11 connector, documentation and software..

NEW RS232

**$89**
Windows Kit:
RS232, buffered,
+ VB source

**$65**
DOS Kit:
TSR/Polling 'C', BASIC, PASCAL

HC2000 V2.1 TW523
Control language (DOS) **$65**

GO ONLINE FOR FREE

Complete range X-10/Leviton. Universal remotes (RS232), DSC **alarms.**
**VISA/MC/AMEX accepted.** Volume Discounts.

Baran-Harper Group, Inc.
Orders: (800) 661-6508 Fax: (905) 471-3730
Technical: (905) 294-6473 BBS: (905) **471-6776/9574**

#128

Listing 1-continued

```
4020 LINE  (TVPX + (OPX(FP(Z)) *  AR),  TVPZ  OPZ(FP(Z)))-
           (TVPX + (OPX(TP(Z)) *  AR),  TVPZ  OPZ(TP(Z))), 0
4030  LINE  (TVPX + (PX(FP(Z)) * AR),  TVPZ  PZ(FP(Z)))-
           (TVPX + (PX(TP(Z)) *  AR),  TVPZ  PZ(TP(Z))), 1
4040  LINE  (RVPZ + (OPZ(FP(Z)) * AR),  RVPY  OPY(FP(Z)))-
           (RVPZ + (OPZ(TP(Z)) *  AR),  RVPY  OPY(TP(Z))), 0
4050  LINE  (RVPZ + (PZ(FP(Z)) * AR),  RVPY  PY(FP(Z)))-
           (RVPZ + (PZ(TP(Z)) *  AR),  RVPY  PY(TP(Z))), 1
4060  LINE  (FVPX + (OPX(FP(Z)) *  AR),  FVPY  OPY(FP(Z)))-
           (FVPX + (OPX(TP(Z)) *  AR),  FVPY  OPY(TP(Z))), 0
4070  LINE  (FVPX + (PX(FP(Z)) *  AR),  FVPY  PY(FP(Z)))-
           (FVPX + (PX(TP(Z)) *  AR),  FVPY  PY(TP(Z))), 1
4080  NEXT Z

4090  CIRCLE (TVPX + (OCX * AR),  TVPZ   OCZ), 5,  0
4100  CIRCLE (TVPX + (CX * AR),  TVPZ   CZ), 5,  1
4110  CIRCLE (RVPZ + (OCZ * AR),  RVPY   OCY), 5,  0
4120  CIRCLE (RVPZ + (CZ * AR),  RVPY   CY), 5,  1
4130  CIRCLE  (FVPX + (OCX * AR),  FVPY OCY), 5,  0
4140  CIRCLE  (FVPX + (CX * AR),  FVPY CY), 5,  1
4150  GOTO 230
```

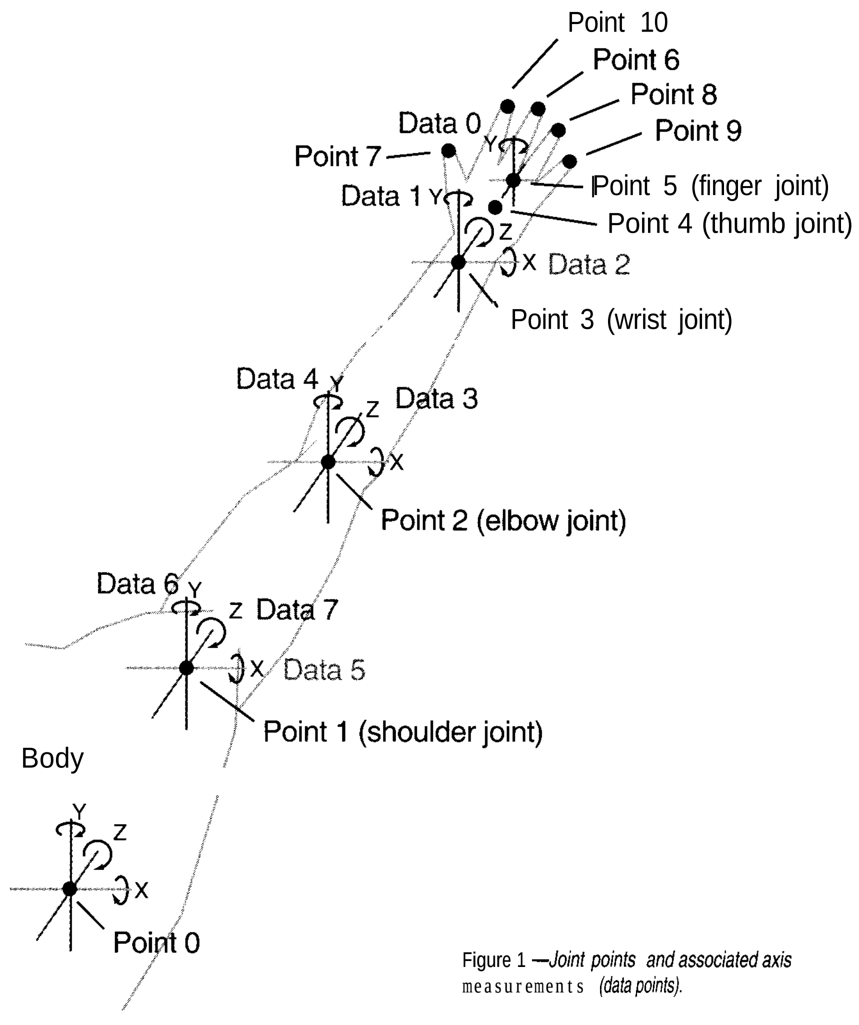


Figure 1 —*Joint points and associated axis measurements (data points).*

length of the lever] x sin(Y-angle) (or sin( Z-angle))

The TVPZ(1) [top view depth] = TVPZ(0) [joint Z-axis point] + L( 1) [full length of the lever] x cos(X-angle) [or cos(Y-angle)]

See the top view in Figure 3 for the drawing showing this point of view.

As each new lever and joint is added, a new point is created in the space that is based on the present location of the last point. The new point's X-, Y-, and Z-angles are relative to all the previous points going back to the base (center of the universe). In this way, new points are affected only by older points. For example, if only the wrist joint moves, the fingers move with the wrist, but the elbow does not. However, if the elbow moves, so do the wrist and fingers.

Computing a new point's position is based on the previous point's position as the base to the new location's relative offset direction. An offset of 0 (on all axes) just continues in the same direction as that of the previous point.

Once the polar angle positions are calculated and stored in the polar joint array for all the joints (points), they are converted into a linear coordinate system so the resulting X, Y, and Z components can be used to visualize the new position (and relative movement) on the PC screen.

The linear coordinate system consists of three axes (as in the polar coordinate system), but instead of measuring each axis's rotational angle to the point in space, the measurement consists of how far along the axis the point is from the center to where the point creates a right angle with the axis. Each polar coordinate point identified by an X-, Y-, and Z-angle from the center of its universe is converted to a linear coordinate identified by ±X-, Y-, and Z-axis distances from the same center of the universe. See Figure 4.

## PC PROGRAM

The PC's BASIC program begins with an opening of a COM port for data reception, followed by an account of all variables used. Included here are the X, Y, and Z points (which are both joints and the ends of connection levers]. A second set of these points is also kept: the first set describes the new positions (for drawing) and the second set describes the old positions (for erasing). In addition, three sets of X-, Y-, and Z-axis angles are kept for each point: the original X-, Y-, and Z-axis angle; the joint's present X-, Y-, and Z-axis angle; and the total for each axis up to this joint. Finally, two link variables are used. These are responsible for linking the X, Y, and Z points in the right order to build the model.

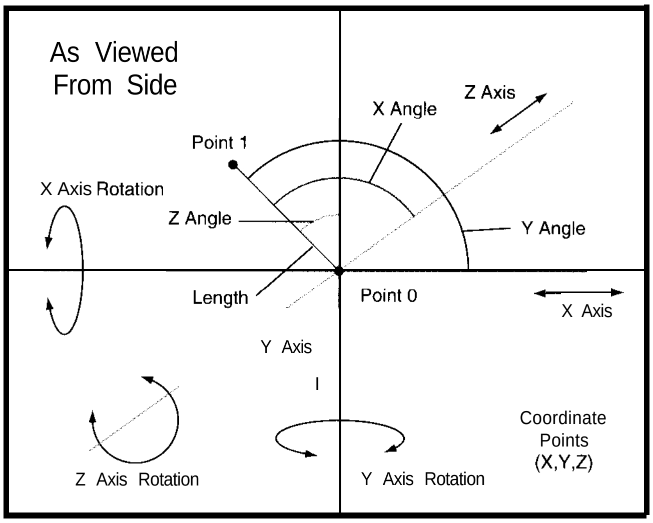


Figure 2—*The first calculations determine the position of each joint in fhe universe and are derived using a polar coordinate system.*

Each point is defined by an array number; all the variables associated with that point have the same array offset. Some of these are predefined; for example, which points will be connected by line segments, what are the initial X-, Y-, and Z-axis angles for each point, and what is the length of each line segment (body part].

The screen is put into 640x200 graphics mode and divided into four quadrants. The upper left is labeled "Top View," the lower left is the "Front View," and right side holds the "Right View." The top right quadrant is used to list items of interest (e.g., the eight incoming data angles produced by the exoskeleton).

At this point, the program enters an endless loop of collecting, computing, and displaying data. Five routines are used for this function. The first routine signals the external micro to send the eight data variables and then stores them in an array. The second routine saves the present X, Y, and Z points in the variables used in the erasing routine. The third routine updates the total axis angle (for X, Y, and Z) for each joint from the original starting angle, the present data angle, and the total angles of all the previous joints. The forth routine computes new joint locations based on the X-, Y-, and Z-axis angles' totals and the length of the line segment. Lastly, the three old views are erased by redrawing their line segments with the background color and the new views are drawn using the foreground color.

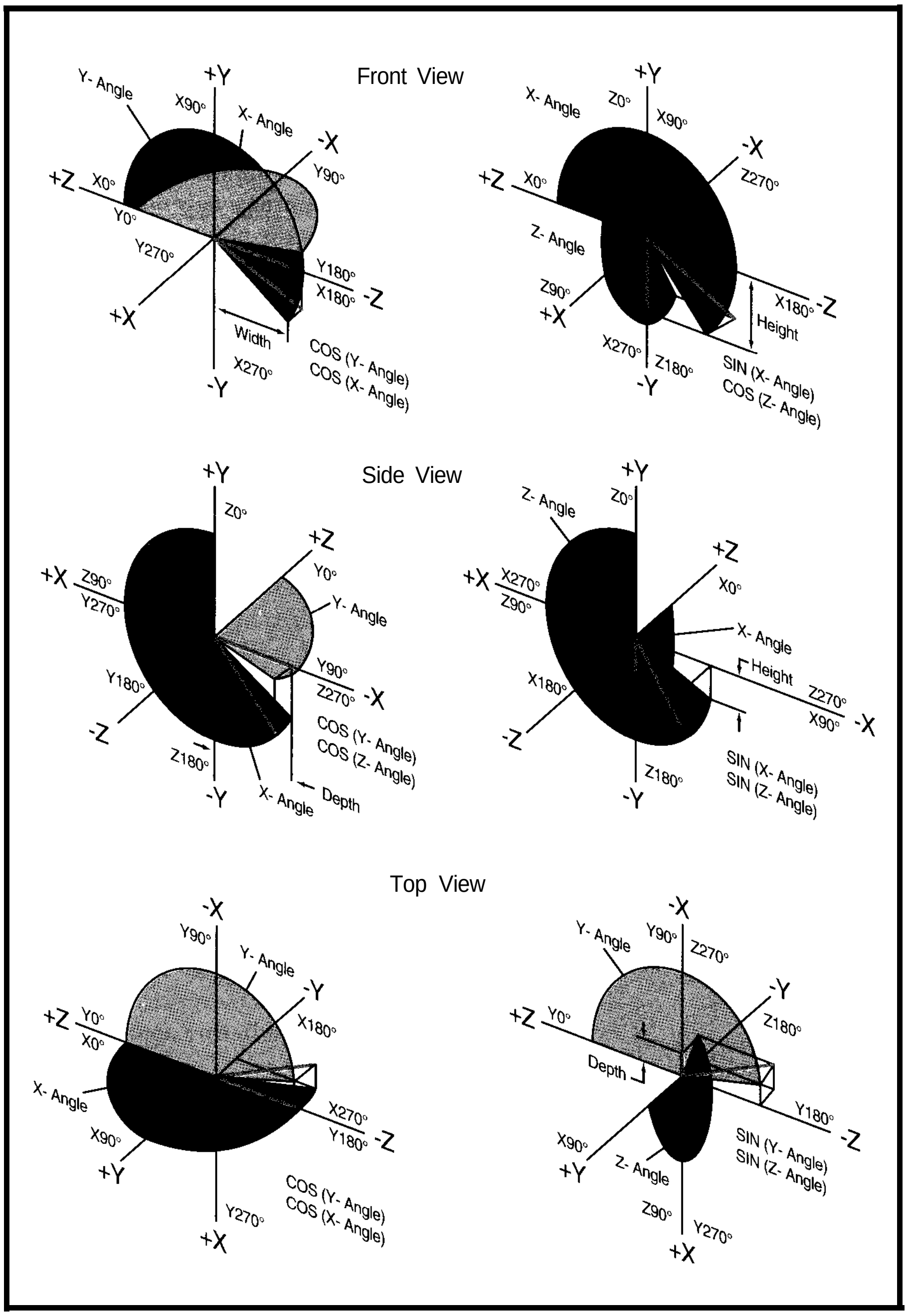


Figure 3—*By placing a joint at the center of fhe universe, the movement of a second joint pivoting about the first affects what is viewed from three perspectives: front, side, and top.*

For those of you who have not built the exoskeleton, I included a routine to enter joint movements via your keyboard. Answering the opening question, "Are you using the exoskeleton?" with "N" allows external data to be produced using the "1-8" keys. Each press of a key will increment that joint's angle data by 10" [holding the shift key while pressing a number will decrement it by 10°).

Some people won't consider this to be real VR because there is nothing in the virtual world to react with. For you, I've added a little something extra. See Photo 2. The arm can be used to pick up, move, and place a ball anywhere on the screen. But please be careful: you can also place it off the screen. I've lost more balls in my monitor than I care to admit.

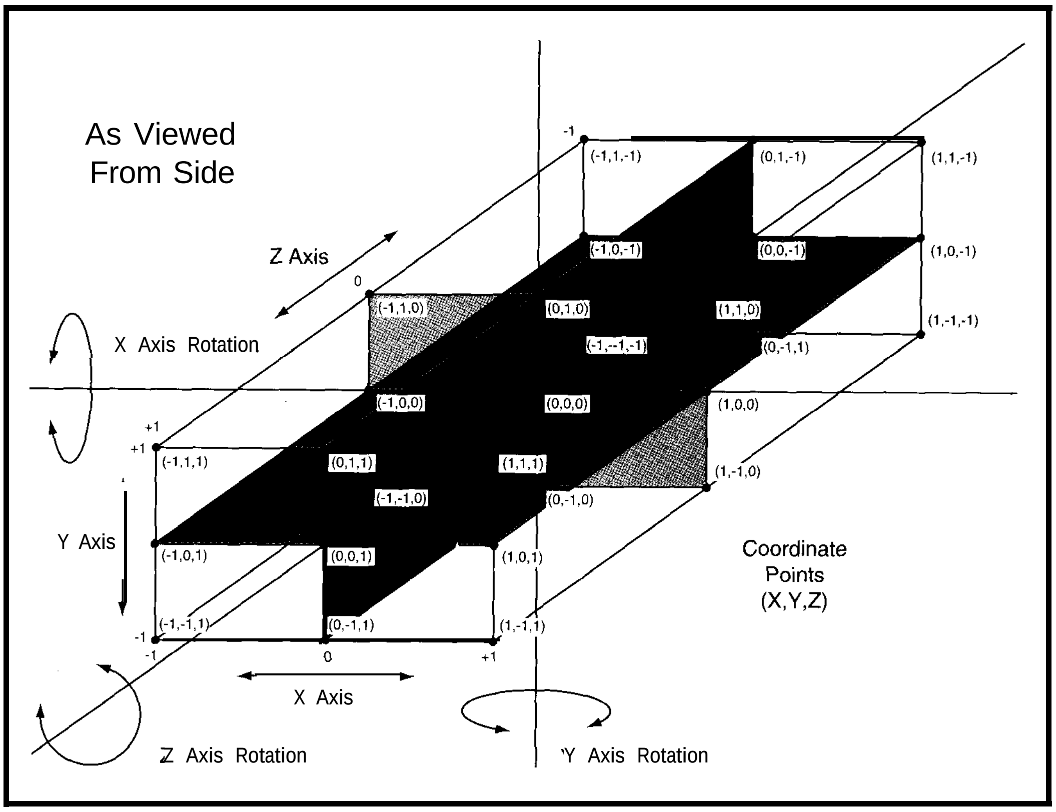


Figure 4-For simplicity, only integer points are shown. However, *the* actual *X,Y,* and Z coordinates can be any mixed number.

Screen updates are produced about once a second on slow machines, but this is based on displaying items of interest as well as movement updates.

**Some** corners have been cut to reduce computation times to a minimum since this is where most of the time is spent. If interest persists in the exoskeleton data input system, I could be coaxed into developing a driver for REND386.

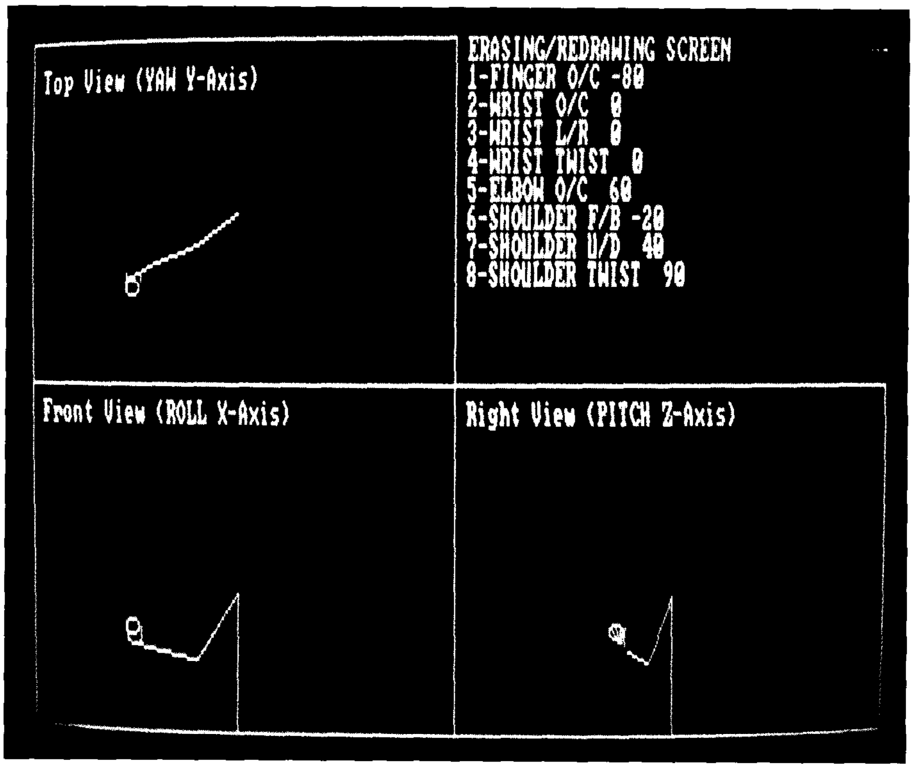

```
Top View (YAW Y-Axis)        ERASING/REDRAWING SCREEN
                             1-FINGER O/C -80
                             2-WRIST O/C  0
                             3-WRIST L/R  0
                             4-WRIST TWIST  0
                             5-ELBOW O/C  60
                             6-SHOULDER F/B -20
                             7-SHOULDER U/D  40
                             8-SHOULDER TWIST  90
Front View (ROLL X-Axis)     Right View (PITCH Z-Axis)
```

Photo **2**—*The* exoskeleton *support software handles simple virtual reality by allowing you to manipulate a ball on the* screen using *the* exoskeleton device.

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

**416** Very Useful
417 Moderately Useful
418 Not Useful

# IrDA—The IR Babel Buster

## SILICON UPDATE

**Tom Cantrell**

What good is the infrared interface on your newest palmtop or PDA when it can't be used to talk to other people's units? A consortium of manufacturers agrees, and the result is IrDA. Find out more.

Infrared surely has an important role to play in the forthcoming "wireless revolution." The technology, thanks largely to the widespread use of "remotes" in the consumer electronics sector, is both proven and cheap.

Yet, it's exactly the bunny-like proliferation of the little beasts that gives pause. How many of you struggle with a quiver of handhelds to manage your entertainment center? Yes, it's not hard to conjure up a grim version of the IR future.,

Tom: Hi, I'm Tom.
***Bill:*** Glad to meet you. I'm Bill, let's exchange business cards.
Tom: OK. I'll show you my PDA if you show me yours. Ho ho!

{The carrier detect light flickers, but won't stay on)

***Bill:*** Hmm, must be a problem. Let's move away from this window.

(They move away from the window, and then-with the carrier detect still flickering-perform kind of a fencing match, each trying to aim their PDA at the other's "sweet spot")

Tom: Oh, I know, I'm still running my TV protocol from last night.
***Bill: No*** sweat, I've got a 950-nm adapter in here somewhere.

{Bill fumbles in his briefcase to extricate the adapter from a tangled deck of PCMCIA cards}

***Tom:*** Got it. Ain't technology grand.

## IRDA TO THE RESCUE

Fortunately, there's an IR compatibility alliance on the horizon: The Infrared Data Association, more affectionately known as IrDA.

The membership list is a Who's Who of system and silicon suppliers. Most notably, it includes existing IR players like HP, Apple, and Sharp who, until now, have been promoting their own proprietary offerings.

Now a gee-whizzy membership list isn't enough to guarantee a hopeful standards success-the ash heap of computing history is littered with a plethora of standards hopefuls that, although "blessed" by leading players, still didn't make it. Even when a standard succeeds, it's often a rather tortuous process. Witness SCSI-2, which, after many years of hem and haw, still isn't officially final. Considering buying one of the new "fast" (100-Mbps) Ethernet cards? Watch out, because there are two "standards" on the shelf [essentially the IEEE 802 committee version and the HP version) and they don't talk to each other.

Maybe this time. The IrDA members are all really true believers in doing the "right thing," even if it's their own ox getting gored. Or maybe it's just battle fatigue, with RISC versus CISC, Windows versus Mac, and the other religious wars having taken the air (mostly hot) out of would-be standards warriors.

Whatever the reason, it looks like IrDA is proceeding toward a final standard at a relatively blistering pace. Notably missing-at least on the surface-is the infighting, back stabbing, and self-dealing that usually characterizes such efforts.

So start looking for the IrDA logo (Figure 1) on any gear you buy or risk being the lonely wallflower at the IR party.

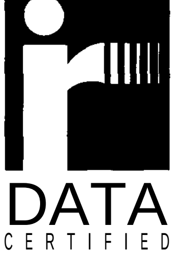


**Figure 1**—*The IrDA logo should soon begin appearing on your favorite portable computing equipment.*

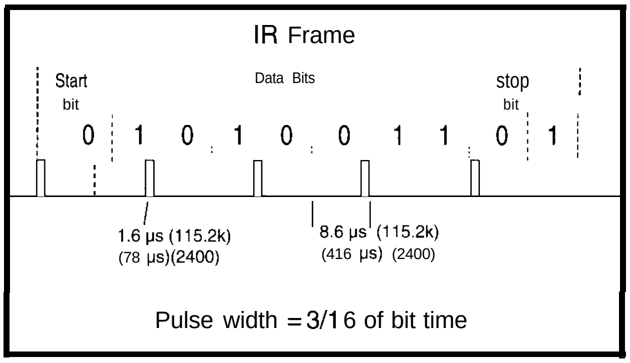


Figure 2—*Though the IrDA format is similar to that used by conventional UARTs, it instead uses short pulses to save precious battery power.*

## I'M OK, UART OK

One factor in the smooth acceptance of IrDA may be that, unlike more grandiose efforts, the scope of the standard is strictly limited to "walk up" (i.e., close-proximity, line-of-sight, point-to-point) data transfers. It doesn't address the "diffuse" IR "wireless-LAN" schemes that are designed to bathe an entire office and must support many-to-many communications. Lovers of chaos will find all they need in the diffuse IR arena, which really needs an IrDA-like standards effort of its own.

The basic parameters-speed and distance-of the IrDA specification scheme were dictated by the key constraints of the portable equipment folks, namely low cost and low power.

Thus, the distance between transferees is limited to O-l meter, though manufacturers can support larger spans (e.g., up to 3 meters) at their discretion. Interestingly, the "zero meter" part was driven by a human factors issue: when a transfer isn't going well, the first thing Joe/Jane Consumer will do is poke their gizmo right up against the balky recipient. As you'll see, this has nontrivial design implications—specifically the need for some kind of variable threshold automatic gain control.

Refreshingly for a business in which NIH always lurks nearby waiting to pounce, the IrDA folks adopt a simple UART scheme for signaling. For most of you this is old hat by now; S-bit characters are transferred least-significant bit first, framed by start and stop bits, and so on. Note that IrDA links are defined as half-duplex (a device can either talk or listen, but not both at the same time). There's no grand strategy behind this, it's simply necessary lest reflections cause a device to end up talking to itself.

The UART heritage is further seen in the IrDA transfer rates of 2400, 9600, 19.2k, 38.4k, 57.6k, and 115.2k, which are the usual rates supported by a PC serial port. Yeah, it doesn't sound like much when everyone is hyping Mega and even Giga goodies, but just how much bandwidth do you need to transfer a business card anyway!

The wide range (almost 50: 1) of transfer rates reflects the scope of devices addressed by the standard. A data rate of 2400 bps may be all that a tiny 4-bitter can bit bang, while 115.2k is where most PCs poop out.

Though UART-like, there are a few subtleties as shown in Figure 2. Notably, the UART's normal "level" output is changed to a "pulse" that occupies only a fraction of the bit time. Obviously, the idea is to minimize the IR LED's on time and, thus, power consumption. In principle, this demands more accuracy from the bit sampling logic, but in actual practice the accuracy of UART clocks—typically crystal controlled-eliminates the concern. Besides, most UARTs actually sample only once in the middle of the bit cell anyway, so why keep the LED on the entire time?

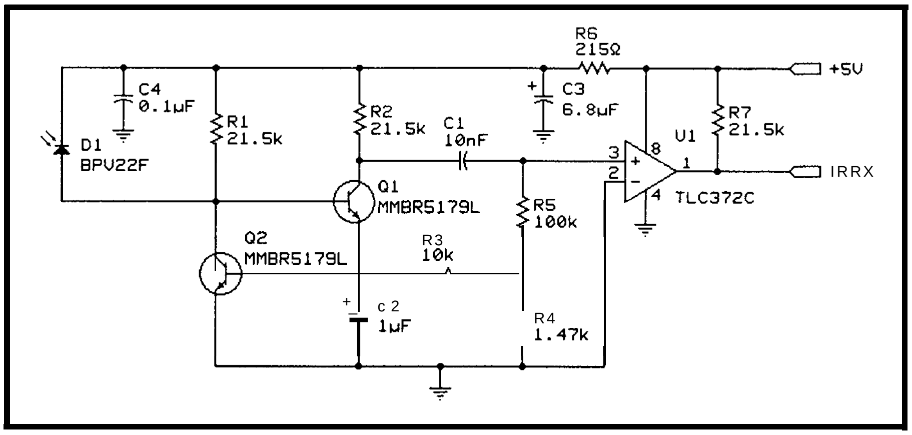


Figure 3-*California Wireless uses a front-end signal conditioning scheme to clean up any unwanted signals.*

IrDA defines the LED on time as between 1.6 µs and 3⁄16 of the bit time. The 1.6-p spec minimizes power consumption, but may be too quick for lowest-cost implementations to deal with. They can fall back on the more leisurely 3⁄16 option.

Power consumption is heavily (inverse squared law) related to distance, a factor behind the minimum l-meter and the optional 3-meter compromise. However, the typical speed/power relationship doesn't hold

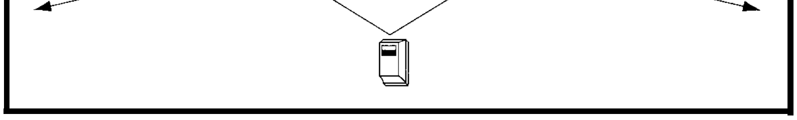


Figure 4-*Without some kind of communications protocol, a scenario in which three (or more) devices are within view of each other could cause problems. IrDA uses HDLC to combate the problem.*

when it comes to the transfer rate. In the case of a fixed pulse width design, the data rate impacts only the time between pulses, not their duration. In other words, the total LED on-time will be the same for a given data transfer, whatever the data rate.

The variable pulse width ($^3/_{16}$ of bit time) is even more paradoxical, in that LED on-time for a given data transfer will increase linearly as the inverse of the data rate. In other words, it takes less power to go faster. Thus, a power-reducing design goal is to wring out as many bits per second as you can.

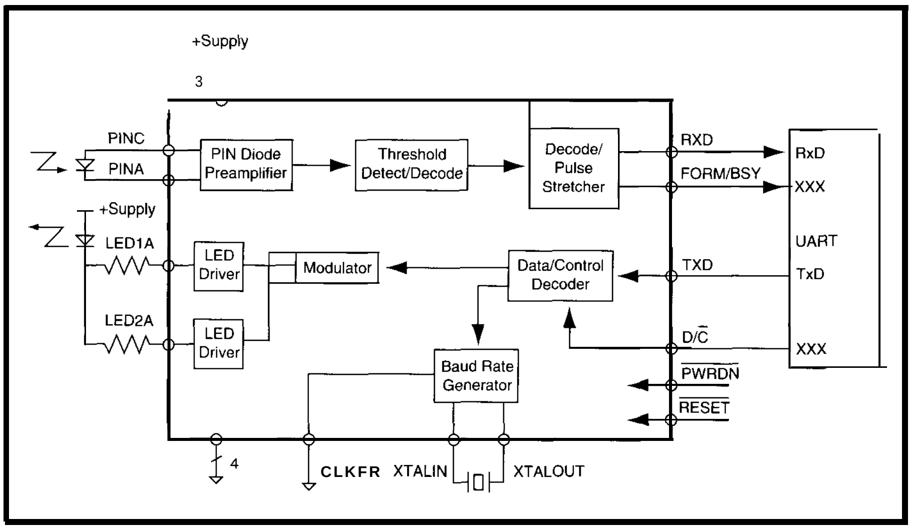


Figure 5—The Crystal Semiconductor CS8130 supports both IrDA format and typical television-type IR formats.

## BLINDED BY THE LIGHT

Compared to an RS-232 connection with nice thick wires and two-digit voltage swing, bits hitchhiking on photons face a much rougher ride.

Besides the fact the signal level varies with distance and orientation, unexpected turbulence is likely to occur thanks to interference from a variety of light sources that contain IR components. Sources of interference include sunlight, incandescent lamps, and switched fluorescent lights.

Worse, assuming IR migrates along with PCs and PDAs into the home, what about all the other consumer IR gadgets such as TV remotes and wireless headphones? While IrDA calls for an 880-nm wavelength, that's pretty darn close to the 950-nm bouncing around your den.

A reference design by California Wireless (Photo 1) shows the type of front end signal conditioning (Figure 3) required to clean things up. Comparator U1 serves as a variable threshold detector to deal with DC bias that determines whether the environmental IR ambiance is bright or dark. The threshold setting is key to the tradeoff between accuracy and distance. If overly sensitive, lots of false zero bits will be detected. If too deadened, transmissions will get lost at ever shorter distances. The rest tackles switched interference by implementing a bandpass filter to reject low-frequency (60 Hz) hum and high-frequency noise.

The IrDA technical committee has performed a variety of reliability tests and for the most part, results are quite good.

One question is whether IrDA can drive your consumer electronics gear crazy. Imagine you're working on the computer at home and start an IrDA transfer. All of a sudden, your TV turns on and switches to "Laverne & Shirley" reruns. Worse, you can't change the channel, lower the volume, or even turn the thing off! Fortunately, testing shows that television IR receivers seem quite immune to IrDA.

The other side of the coin is whether rogue IR can disrupt an IrDA transfer. While it's probably a good idea to filter sunlight, typical home lighting conditions are no threat to the standard's $10^{-9}$ Bit Error Rate spec.

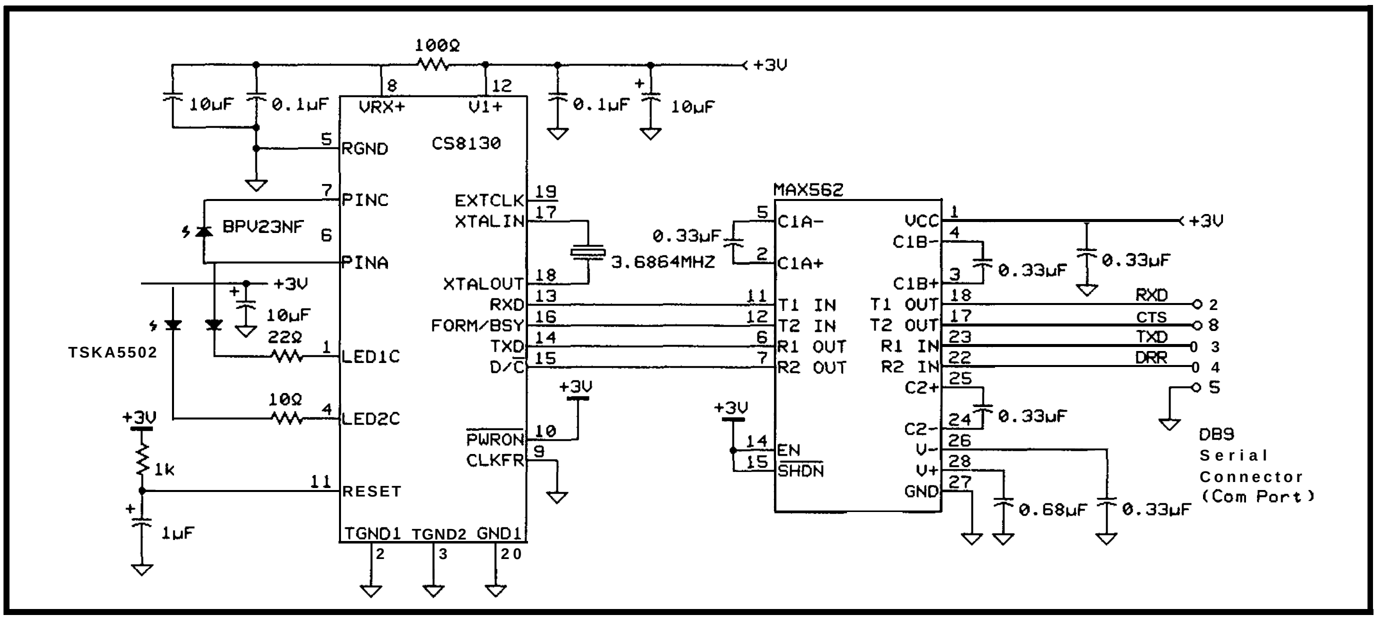


Figure 6—The CS8130 EV board provides instant IrDA with all the accompanying components.

However, testing did show it's possible to glitch a transfer if you blast your TV remote right at an IrDA receiver. Fortunately, higher layer software protocols can provide error recovery to protect even the most rabid channel surfers from shooting themselves in the diode.

## WHO'S ON FIRST

Speaking of software, IrDA is also defining the IRLAP (IR Link Access Protocol) to establish and ensure the integrity of communication links.

Though ostensibly a simple "point-to-point" link, it's clear that IR isn't as pointy as a good old-fashioned wire. Consider the situation in Figure 4. Obviously a scheme is needed to figure out who's talking to whom, lest everyone talk at once or a device end up talking to one partner and listening to another. Also, whether due to low battery or the shaky aim of an aging society, the protocol must handle error detection and recovery.

Rather than reinventing the wheel, IrDA has adopted an HDLC-like (High-level Data Link Control) protocol. HDLC has been around for many years, notably serving as the basis for mainframe (remember them?) communications. Perhaps its most popular incarnation these days is as the core of Apple's LocalTalk.

The key points of HDLC are that it is a primary/secondary scheme in which multiple secondary devices can communicate with a single primary station that's in charge of who gets to talk and when. Transfers occur in packets that consist of address information, data, and CRC. In fact, there are three distinct packet types: U (Unnumbered), S (Supervisory), and I (Information).

Resolution of situations like Figure 4 is handled by an initialization and a "discovery" process that's kind of like a bunch of yuppies networking: "Who are you! What can you do for me? Glad you met me." Everyone starts at 9600 bps and then negotiates for a higher data rate.

## ANTE UP

Before you fire up your soldering gun, be warned that to drive on the

**PIC16C5x** Real-time Emulator

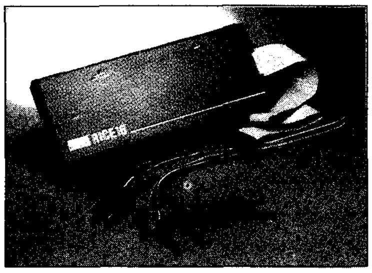
Introducing RICE%-5x and RICE5x-Junior, real-time in-circuit emulator5 for the PIC16C5x family microcontrollers: affordable, feature-filled development systems from **$599***

• Suggested Retail for U.S. only

RICE16 Features:

Real-time Emulation to 20MHz

- 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 Breakpoints
- 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 patching instruction



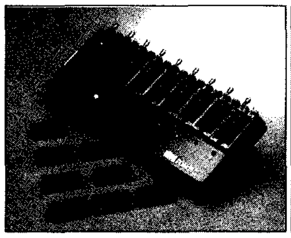
- Support 16C71, 16C84 and 16C64 with Optional Probe Cards
- Easy-to-use Windowed Software
- Comes Complete with TASM16 Macro Assembler, Emulation Software, Power Adapter, Parallel Adapter Cable and User's Guide
- 30-day Money Back Guarantee
- Made in the U.5.A.

RICE5x-Junior supports PIC16C5x family emulation up to 20 MHz. It offers the same real-time features of RICE16 without the real-time trace capture.

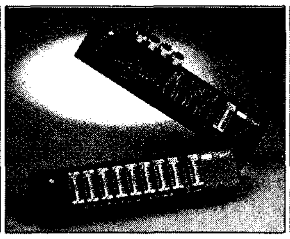
**PIC** Gang Programmers

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

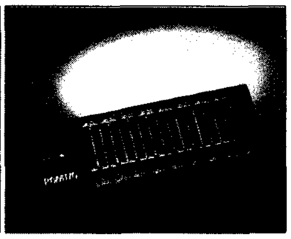
■ Stand-alone COPY mode from a master device ■ PC-hosted mode for single unit programming • High throughput ■ Checksum verification on master device ■ Code protection ■ Verify at 4.5V and 5.5V ■ Each program cycle includes blank check, program and verify eight devices ■ Price5 start at **$599**



PGM16G: for 16C5x family
Throughput: 16C54: 4 seconds
(8 devices) 16C57: 10 seconds



PGM47: for 16C71/84
16C71: 10 seconds
16C84: 15 seconds



PGM17G: for 17C42
17C42: 13 seconds

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 (2141999-2999
14330 Midway Road, Suite 104. Dallas, Texas 75244 Fax (214) 980-2937

#131

IrDA highway, you've got to get a license.

Turns out that the IrDA is largely based on HP's SIR (Serial Infrared) scheme (used in products like the HP95LX and Omnibook), a technology for which they have certain patents.

Lest you protest too much, my impression is that HP is being "forced" to make people pay in order to ensure the "validity" of their patents. It's only fair to point out that HP didn't start the intellectual property arms race and they won't finish it. For good or for bad, it appears the days of "free" standards are over.

Fortunately, you can skip the legalities and simply buy a chip from one of the IrDA roster's many IC suppliers, a good example being the Crystal Semiconductor CS8130 shown in Figure 5. Besides avoiding "party of the first part" heartburn, an integrated IC eliminates the need to fuss with all those resistors and caps. As shown in Figure 6 (the Crystal CS8130 EV board), simply add LEDs, PIN photodiode, and a crystal—voilà, instant IrDA.

The CS8130 is set up (e.g., data rate select) via TXD when the D/C * [Data/Command) input (typically connected to a modem control signal) is driven low. Otherwise (D/C* high], RXD and TXD simply pass data transparently between the computer and IR sides. The FORM/BSY output has dual use, indicating the IR format (the chip supports both IrDA and TV formats] or busy handshaking. The XTALIN and XTALOUT pins connect to either a 1.8432.MHz (16×115.2K bps) or 3.6864-MHz (depending on the CLKFR strapping] crystal.

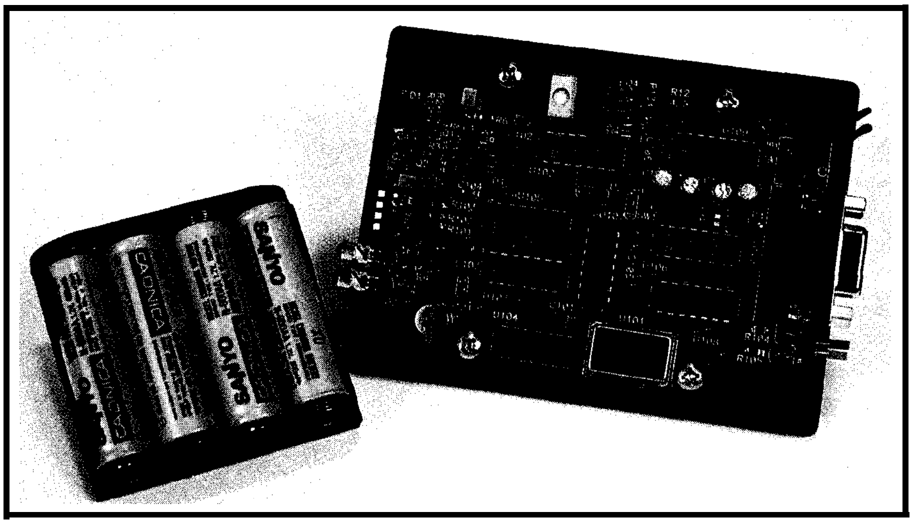


Photo 1—*California* Wireless *IrDA reference* board.

Dual outputs can drive two LEDs or a single LED at four levels determined bv the current-limiting resistors. The TSHA550 GaAlAs, 875-nm, IR LED from TEMIC is a good choice, consuming on the order of 100 mA @ I.5 volts. At only $0.31 in thousands, you can certainly afford to use two in the equivalent of a car's high-beam / low-beam configuration. A smart transmitter can "hit the brights" for maximum visibility.

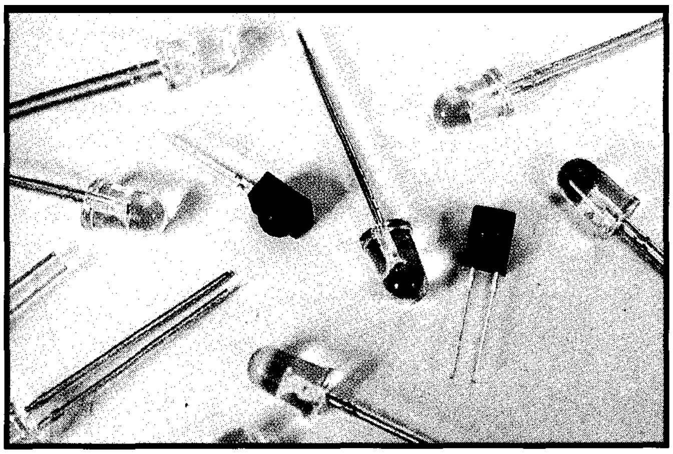


Photo *2-The TEMIC TSHA550 GaAlAs 875-nm IR* LED *and BPV23NF photodiode are perfectly matched and ideal for use in IrDA interfaces.*

On the receiving end, TEMIC offers a well-matched mate in the BPV23NF photodiode (Photo 2) which features a spherical lens for increased sensitivity (quoted as 80% better than a flat package) and wide (±60°) viewing angle. Better yet, it's only $0.37 in thousands.

At the time of this writing, the CS8130 price wasn't final, but you can probably put together an entire IrDA setup-IC, crystal, and diodes-for less than $5.00 in moderate volumes. With increased volume and competition, the cost will likely fall to a couple of bucks, certainly making IrDA a must for any well-connected communicator. ❑

*Tom Cantrell has been an engineer in* Silicon Valley for *more than ten years working on chip, board, and systems design and marketing. He can be reached at (510) 65 7-0264 or by fax at* (510) 657-5441.

## CONTACT

California Wireless, Inc.
618 Enos Ct.
Santa Clara, CA 9505 1
(408) 249-7300
Fax: (408) 249-7301

Crystal Semiconductor Corp.
P.O. Box 17847
Austin, TX 78760
(512) 442-7555 x3253
Fax: (512) 445-7581

Infrared Data Association (IrDA)
P.O. Box 495
Brookdale, CA 95007-0495
(510) 943-6546
Fax: (510) 934-5241

TEMIC Customer Development
2201 Laurelwood Rd.
Santa Clara, CA 95054
(800) 554-5565 x25
Fax: (408) 970-3950

## I R S

419 Very Useful
420 Moderately Useful
421 Not Useful

# Add Some Code to the BIOnet

EMBEDDED TECHNIQUES

John Dybowski

Networking hardware is pretty worthless without some control code. John brings the BIOnet to life with a mix of C and assembler routines.

Last month I covered some of the fundamentals behind the BIOnet binary network. This control/sense network would most likely form the basis of a fixed-base premises monitoring system. It could function equally well as the backbone of an industrial equipment and apparatus monitoring system. Although a variety of ways exist to acquire binary data without the use of any wiring-RF, IR, or even power line-it is well known that wired communications offers higher levels of security when supervising critical operations. The BIOnet provides the advantages of a hardwired system while dealing with some of the difficulties associated with configuring a wired system. This is accomplished by providing bidirectional communications and power over a single, low-cost, twisted-pair cable.

A standard configuration accommodates up to 30 outlying satellite units for a total of 60 inputs and 60 outputs. Applying a slight twist to the technology, the BIOnet is supported by the compact ec.25 Data Collection Computer and can function as a data acquisition subsystem. Combined with the local digital I/O, analog I/O, RS-485 and RS-232 serial I/O, and user I/O via keypad and display, the ec.25 stands ready to function as a complete portable (or stationary) data collection and processing center. And being optimized for low-power operation, the ec.25 is capable of operating from line power with battery backup or from just battery for extended periods.

Since I was running out of space last month, I had to start off with a description of the second-level processing that supports the BIOnet. This discussion took hardly any space at all since the second level operates as the intermediary between the application and the driver. It manipulates data and hands off the real work to the BIOnet driver code. This driver, shown in Listing 1, is written entirely in assembly language and is where the line communications are actually managed. Hopefully by the time I conclude this column, it will be apparent why I selected assembler as the language of choice for coding such bit-intensive operations.

Essentially, the driver program has two user-callable entry points. As you would expect, these perform the two most essential operations associated with network communications: reading from and writing to a single satellite module. BIOnet communications are carried out over a single twisted-pair cable that is also used for supplying power to the network constituents. In this scenario, the processor has the capability of injecting 15 V and 0 V onto the line by turning on one of two control bits. These bits drive either a high-side or a low-side switching transistor into saturation, providing nearly a full-rail swing. The intermediate 7.5-V level is established via a resistor divider and is the default condition when both transistors are off. The line is monitored using a simple inverting voltage comparator that switches at about a 5-V threshold. If the line voltage exceeds this threshold, it is interpreted as a "one" bit; if it is less than 5 V, it is a "zero" bit. It is just through the use of these two output and one input bits that communications with the satellite modules is maintained.

There are different ways you can go about writing peripheral drivers. Sometimes circumstances permit you to code the functions as you conceptualize the algorithms in your mind. Then there are those cases where the implementation details bear little resemblance to the underlying organization. There are many valid reasons for departing from rational coding techniques, but usually the driving force is some pragmatic constraint.

Listing 1—The *BIOnet driver code* is where the *real work* happens in communicating *with remote modules.*

```
;GLOBAL ENTRY POINTS
        PUBLIC EnableBio
        PUBLIC DisableBio
        PUBLIC BIO_READ
        PUBLIC BIO_WRITE

;EXTERNAL REFERENCES
        SENSE   equ     p2.0
        BIO_EN  equ     p2.1
        L15     equ     p2.2
        LO      equ     p2.3

;ASSEMBLEINTO CODE SEGMENT
        PROG    SEGMENT CODE
        RSEG    PROG

;DELAY ROUTINE FOR 10-kHz SCAN RATE

DELAY_25US:
        MOV     R1,#12
        DJNZ    R1,$
        RET

DELAY_50US:
        MOV     R1,#24
        DJNZ    R1,$
        RET

;TRANSMIT A ONE BIT OVER BIOnet
XMIT_ONE:
        SETB    LO
        CLR     L15
        CALL    DELAY_50US
        SETB    LO
        SETB    L15
        CALL    DELAY_50US
        RET

;TRANSMIT A ZERO BIT OVER BIOnet
XMIT_ZERO:
        CLR     L15
        CLR     LO
        CALL    DELAY_50US
        SETB    LO
        SETB    L15
        CALL    DELAY_50US
        RET

;TRANSMIT SYNC PATTERN OVER BIOnet
XMIT_SYNC:
        CALL    XMIT_ZERO
        MOV     R0,#8
XSS1:
        CALL    XMIT_ONE
        DJNZ    R0,XSS1
        CALL    XMIT_ZERO
        RET

;TRANSMIT BIT FIELD OVER BIOnet
;INPUT: ACC CONTAINS BIT PATTERN
;       R0 CONTAINS BIT COUNT
XMIT_FIELD:
XTT0:
        RRC     A
        JC      XTT2
XTT1:
        CALL    XMIT_ZERO
```

(continued)

Often, program or data memory limitations tend to cramp your style, or you may be approaching your processor's performance limits.

When faced with such formidable constraints, engineers usually have no choice but to fall back to less-than-structured programming techniques. Ideally, it's best to partition your code among several layers of virtual processes. This tends to distribute the complexity and permits you the luxury of incrementally developing and debugging the algorithms. As usual, it pays to consider the overall issues before getting too far along. This is especially true when working with some of the popular 8-bit processors that are actually single-chip microcontrollers with external memory added.

The first thing to go in such an arrangement is stack space since it is often implemented in on-chip memory and therefore is of fixed and limited size. With heavy interrupt activity, particularly with multiple levels of interrupt priorities, you could easily swamp the stack region if the mainline code is layered too heavily. And when it's necessary to squeeze maximum performance out of a minimal processor, the first thing you usually want to do is limit your stack manipulations and the associated overhead.

The end result of this brute-force approach to performance enhancement is straight-line code. Although such coding needn't be totally unstructured, and if done properly can actually prove to be quite intelligible, but it does fall short of the ideal. The fact is, when using economical processors, you have to know when it's appropriate to apply what academic types would tell you are inappropriate programming techniques. Fortunately, this is not the case with the BIOnet driver I'm going to show you. Consequently, I will proceed by starting with the smallest pieces and working upwards.

Sometimes you just can't seem to be fast enough no matter what you do. Then there are times when the situation is entirely different and you find you've got time to burn. Communications over the BIOnet can be carried out at up to 20 kHz. Due to the simple electrical line driving scheme I am running, I find I can squeeze a little more distance out of the network by running it at a more pedestrian 10 kHz. With such low throughputs, it should be no problem sustaining the maximum 20-kHz data rate even using a rather anemic processor. Rather than waste time running delay loops or executing endless sequences of NOPs, it's much better to take advantage of those extra clock cycles to render the code in as straightforward and comprehensible manner as possible.

Having established that a 10-kHz data rate is adequate for our purposes, the first thing to do is to define a couple of time delay routines in order to fill out a half-bit time and a full-bit time. Precision is not a requisite function as far as BIOnet communications goes, so no attempt is made to tune the delay loops to compensate for the surrounding code. Actually, the specified satellite communications rate extends all the way from 20 kHz down to DC. Note these figures strictly define the communications parameters and not the system parameters. Holding the line low at 0 Hz would obviously deprive the network of power in short order since the data line is rectified and filtered to provide power at the satellites.

The next couple of routines are designed to transfer the smallest communications component: a data bit. Pulling the line from 7.5 V to 15 V produces a "one" bit and taking the line from 0 V to 15 V generates a "zero" bit. The simplest bit stream consisting of "one" and "zero" bits is the sync pattern. Prior to the actual transfer of information over the BIOnet, synchronization with the network satellites must be established. This is done by sending a "zero" bit followed by eight "one" bits followed by a final "zero" bit. XMIT_SYNC performs this service using just a few lines of code now that the supporting functions have been established.

Comprehensible communications require the transfer of varying bit patterns of variable size. These fields are the constituents of data packets and are used to convey information such as the satellite address, the read or write operation, and the actual data

Listing 1-continued

```
        SJMP    XTT3
XTT2:
        CALL    XMIT_ONE
XTT3:
        DJNZ    R0,XTT0
        RET

;TRANSMIT ADDRESS OVER BIOnet
;INPUT: ACC CONTAINS BINARY ADDRESS
XMIT_ADDRESS:
        ANL     A,#11111B
        MOV     C,P
        MOV     ACC.5,C
        MOV     R0,#6
        CALL    XMIT_FIELD
        RET

;READ BIOnet BITS
;INPUT: ACC CONTAINS BINARY ADDRESS
;OUTPUT:ACC CONTAINS PD,D1,D0 IN 3 LSBS
BIO_READ:
        CALL    XMIT_SYNC
        CALL    XMIT_ADDRESS
        CALL    XMIT_ZERO                ;READ COMMAND
        MOV     R0,#3
RSS0
        CLR     L15
        CALL    DELAY_25US
        MOV     C,SENSE
        CALL    DELAY_25US
        CPL
        RRC
        SETB    L15
        CALL    DELAY_50US
        DJNZ    R0,RSS0
        SWAP
        RR
        ANL     A,#111B
        RET

;WRITE BIOnet BITS
;INPUT: ACC CONTAINS BINARY ADDRESS
;       B CONTAINS D1, D0 IN 2 LSBs
BIO_WRITE:
        CALL    XMIT_SYNC
        CALL    XMIT_ADDRESS
        CALL    XMIT_ONE                 ;WRITE COMMAND
        MOV     A,B
        ANL     A,#11B
        MOV     C,P
        MOV     ACC.2,C
        MOV     R0,#3
        CALL    XMIT_FIELD
        RET


;ENABLE NETWORK POWER
EnableBIO:
        CLR     BIO_EN
        RET


;DISABLE NETWORK POWER
DisableBio:
        SETB    BIO_EN
        RET
        END
```

to be transferred. XMIT_FIELD accepts a bit pattern in the accumulator with the number of bits to transmit in R0. As the routine proceeds, the bits are tested in sequence and control is handed off to the low-level XMIT_ONE and XMIT_ZERO routines as appropriate. Now that XMIT_FIELD is in place, which is a routine that simply transfers raw data bits, the next level of processing must deal with the actual data content.

XMIT_ADDRESS accepts the satellite address in the accumulator and assumes five significant bits of data. The unused bits are masked off and an even parity bit is generated and appended as the MSB. Before control is transferred to XMIT_FIELD, R0 is set to 6 to indicate the number of bits that are to be transmitted-five address bits and a parity bit.

The functions I've described thus far exist to serve the two main user-callable routines. These are defined as the public BIO_READ and BIO_WRITE entry points. BIO_READ is responsible for acquiring the status of the specified satellite module. This status consists of the state of the two input bits plus the parity bit. Playing it safe, I invoke the synchronization sequence prior to every read or write operation since the condition of the satellites somewhere out on the wire is-in my opinion—always questionable. Although it could be argued that it is wasteful to perform synchronization for every read or write operation, experience proves it to be the prudent thing to do. In any case, if network performance becomes such big issue, you could probably fall back to doing the synchronization once per network pass and still run fine.

The BIO_READ routine takes the S-bit satellite address via the accumulator as its input and returns the satellite status in the accumulator via the three LSBs. Following the issuance of the synchronization pattern, the satellite address is dispatched. At this point, the address has already been properly positioned in the accumulator and control is simply passed to XMIT_ADDRESS.

Next, a "zero" bit is transmitted which tells the satellite that this is a read operation. Immediately following

Listing 2—The BIOnet driver routines in Listing 1 interface to this second-level support code written in C.

```
#pragma LARGE CODE
#include "reg5000.h"

#define Sense       P2_0     /* 1 = 0 */
#define BioEn       P2_1     /* 0 = on */
#define Line15      P2_2     /* 1 = on */
#define Line0       P2_3     /* 0 = on */

/* Delay Routines for 10-kHZ rate */
static void Delay25us(void)

    unsigned char c:

    c = 6;
    while (c--);
    return;


static void Delay50us(void)
{
    unsigned char c;

    c = 14;
    while (c--);
    return:


/* Transmit a one bit */
static void XmitOne(void)
```

(continued)

# Datrax 52 Computer

Remote Terminal - Programmable Controller - Complete System

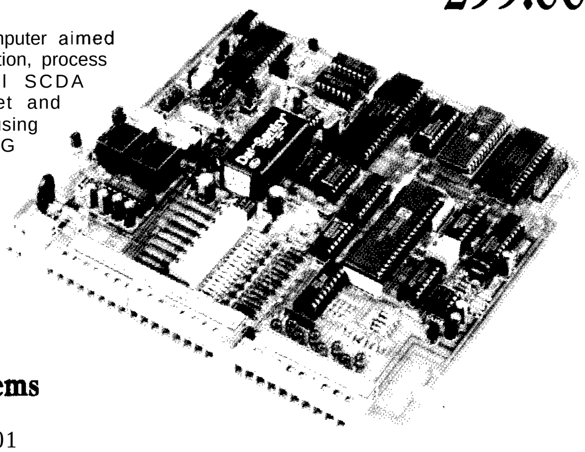
- 80C52 CPU
- High Speed Floating Point BASIC
- 32K Nonvolatile RAM
- 16K EPROM Programmer
- RS-232 Com & Printer
- Watch Dog Timer

- Eight 4-20 mA Inputs
- Four Discrete Inputs
- Four Relay Drivers
- 12 Diagnostic Lamps
- Extensive Surge Protection
- 12/24 Vdc Power

Options:
- Bell Compatible 202 Modem
- Clock/Calendar
- 32 Bit Counter
- 4-20 mA Output

$299.00

The Datrax 52 is a computer aimed primarily for data acquisition, process control, and general SCDA applications. A target and development system using HIGH SPEED FLOATING POINT BASIC on a single board. For a complete description please call or FAX.



**Data Track Systems**
2829 Lewis Lane
Owensboro, KY 42301
Phone (502) 926-0873
FAX (502) 683-9873

the transmittal of this command bit, preparations are made for entry into the read loop. This loop executes three times to read the satellite's two input bits and the parity bit. The loop proceeds by first releasing the line to 7.5 V and, after a short delay, the input bit is read and positioned. The satellite is informed that the bit has been read and that it may stop driving the line. This is conveyed, in a rather blunt manner, by yanking the line up to 15 V. After three iterations of this loop, the code falls through and does the final bit positioning, then clears any superfluous bits, retaining only the three least-significant bits in the accumulator. These three bits and the PSW's parity bit are returned to the caller. Note that if an attempt is made to access a satellite that is not present, an idle line will be read which, by default, returns three "one" bits. This automatically registers as a parity error, which is the desired result.

BIO_WRITE proceeds in a similar fashion, but following the transmission of the address, a "one" bit is sent to indicate that a write operation is to follow. The output bits are pulled out of the two LSBs of B and an even parity bit is appended before XMIT_FIELD is given control to transmit these three bits. Enabling and disabling the 15-V power consists of nothing more than setting or clearing a port pin. For completeness, these functions are performed by a few callable routines.

## IF YOU INSIST

Having coded the BIOnet driver in assembler, it would be instructive to look at how these same functions could be rendered in a higher-level language such as C. Now, some might say that C is not really a high-level language at all and would maintain that it more resembles a stylized assembler in its ability to get close to the hardware. No matter how you rationalize it, the fact remains that some of the language's power and features tend to get in the way when you're trying to do things at the machine level. Although I've managed to go lower using C than I would have thought practical, there comes a point where you definitely begin to work

Listing 2-*continued*

```
{
    Line0 = 1;
    Line15 = 0;
    Delay50us();
    Line0 = 1;
    Line15 = 1;
    Delay50us();
    return:


/* Transmit a zero bit */
static void XmitZero(void)

    Line15 = 0;
    Line0 = 0;
    Delay50us();
    Line0 = 1;
    Line15 = 1;
    Delay50us();
    return:


/* Transmit sync pattern */
static void XmitSync(void)

    unsigned char c:

    XmitZeroO;
    for (c = 8; c != 0; c--)
        XmitOne();
    XmitZeroO;
    return;
}


/* Transmit a bit field */
static void XmitField(char FieldLength, char BitField)
{
    while (FieldLength--){
        if (BitField & 1)
            XmitOne();
        else
            XmitZeroO:
        BitField = BitField >> 1;

    return;


/* Transmit address */
static void XmitAddress(char Address)
{
    ACC = Address:
    ACC &= Oxlf;
    ACC_5 = P;
    Address = ACC;
    XmitField(6, Address);
    return;


/* Read Satellite inputs */
void BIO_READ(void)

    unsigned char c:
    unsigned char Bits;
    unsigned char Address:

    Address = ACC;
    XmitSync();
```

(continued)

Listing 2—*continued*

```
     XmitAddress(Address);
     XmitZero();
     Bits = 0;
     c = 3;
     do {
         Line15 = 0;
         Delay25us();
         if (!Sense)
             Bits |= 8;
         Bits = Bits >> 1;
         Delay25us();
         Line15 = 1:
         Delay50us();
     } while (--c);
     ACC = Bits:
     return:


 /* Write Satellite outputs */
 void BIO_WRITE(void)

     char Address, Bits;

     Address = ACC;
     ACC = B;
     ACC &= 3;
     ACC_2 = P;
     Bits = ACC;
     XmitSync();
```

*(continued)*

against your tools. Or are the tools working against you!

When working with compilers, it's interesting to examine what the code generator has been up to. In some cases, I've been delighted to discover that the compiler had, in fact, actually done what I meant rather than what I said. Unfortunately, there were also those cases where my instructions seemed clear enough to me, yet the compiler decided to generate some diabolically complex sequence that somehow performed what I had intended. It really does pay to analyze the output of your compiler until you get a good feel for the quality of code you're getting. Anyway, there does come a point when you're working on the fringe, and using a code generator becomes somewhat of a force fit. You can tell when you're pushing it when you find yourself expending more effort in coercing your tools than in applying them to getting the job done.

The BIOnet driver interfaces to the second-level support code, which is written in C. Naturally, if you

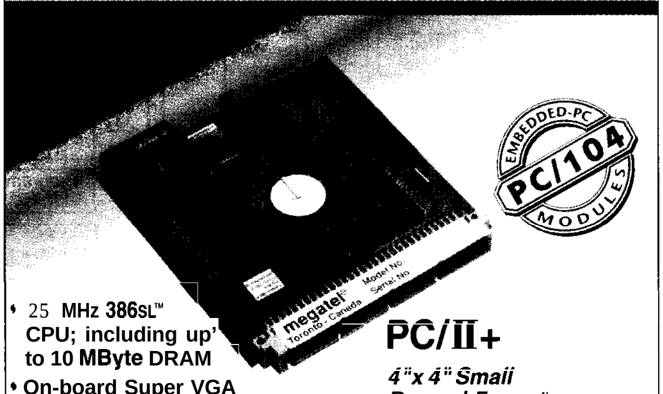
WORKS

Does your Big-Company marketing department come up with more ideas than the engineering department can cope with? Are you a small company that can't afford a full-time engineering staff for once-in-a-while designs?

Steve Ciarcia and the Ciarcia Design Works staff may have the solution. We have a team of accomplished programmers and engineers ready to design products or solve tricky engineering problems. Whether you need an on-line solution for a unique problem, a product for a startup venture, or just experienced consulting, the Ciarcia Design Works is ready to work with you. Just fax me your problem and we'll be in touch.

**Remember...**
*a Ciarcia design works!*
Fax (203) 871-8986

Embedded **PC**
with on-board **Ethernet**
and **Super VGA**



- 25 MHz 386SL™ CPU; including up to 10 MByte DRAM
- On-board Super VGA LCD/Video controller
- On-board Ethernet, Featuring AUI and 10 BASE-T interfaces
- On-board SCSI, Floppy controllers and 2 MByte Flash Eprom Solid State Disk
- 3 Serial Ports, Parallel/Printer port

**PC/II+**
*4"x 4" Small Rugged Format*
For more information call:
Megatel Computer Corporation
125 Wendell Ave., Weston, Ont.
M9N 3K9 Fax: (416) 245-6505
**(416) 245-2953**

Megatel is a registered trademark of the Megatel Computer (1986) Corporation 386SL is a trademark of Intel Corporation

**megatel"**

#132

decided to code the driver in C, you would be wise to use the inherent parameter passing conventions of the language. I didn't follow this practice here since I wanted to keep a one-to-one correspondence between the handcrafted assembler code and the compiler-generated C code. This shouldn't obscure the basic structure since it only involves a little front-end and back-end processing to sort out the passed arguments. Standard parameter passing is used between the local functions since this is primarily an internal issue here.

Looking at Listing 2, you can see that things start out fairly cleanly with the basic time delay functions. Aside from the need to consult the executable code to verify that the proper time delays will be counted, there's not much going on here. Similarly, the bit transmission routines generate code that corresponds very closely to the original assembler. Likewise, the sync and bit-field transmission functions generate some fairly respectable code as well. Looking at the address transmission routine, you start to get an inkling that things are taking a downturn at this point.

All along, I've been using the compiler's special low-level capabilities to get at the DS2250's hardware, but now I find myself forced to dabble at the processor's register level to facilitate parity generation. Things continue to deteriorate with B I 0_ READ, which is one of the main public entry points. In case you're interested, the actual assembler output of B IO_ READ is shown in Listing 3. This

Listing **2**—*continued*

```
    XmitAddress(Address);
    XmitOne();
    XmitField(3, Bits):
    return:


/* Enable BIOnet power */
void EnableBio(void)

    BioEn = 0;
    return:


/* Disable BIOnet power */
void DisableBio(void)

    BioEn = 1;
    return:
```

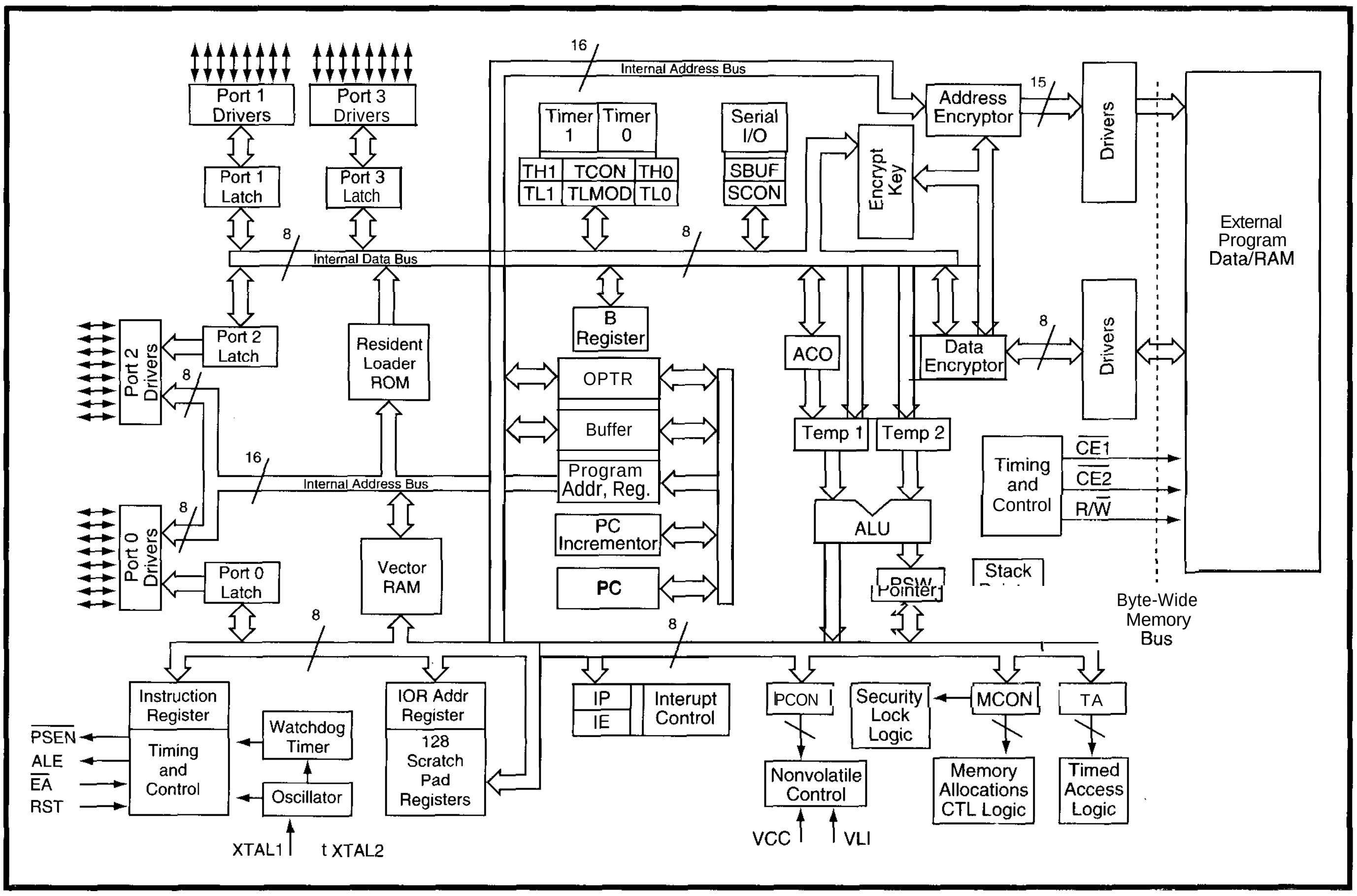


Figure I-While the *DS2250* and the DS.5000 may look similar to the 8031, a peek under the covers reveals a host of new features.

illustrates several important points. First of all, what the code looks like and what it is are often two entirely different things.

Obviously, source code economy is not what we're after, so don't knock yourself out making your source statements so impenetrable that even you can't crack it. Secondly, if timing were an issue here, obviously you'd be in for trouble. Depending on the particular compiler you're using, it may even be questionable whether you can get the low-level control you need for a specific task. To a great extent what makes coding the BIOnet driver in C practical are the compiler-specific extensions that were included in this implementation. To say that this is not portable code would be a gross understatement. The bottom line is that coding drivers of this type in C may not necessarily be in your best interest. Even though the code does in fact do the deed, it takes a little bit to get from here to there. BIO_WRITE, although similar, produces somewhat cleaner output.

Listing 3—*The* assembler *output for BIO_READ illustrates the difference between what the code looks like and what it actually is.*

```
; FUNCTION BIO_READ (BEGIN)

        MOV       R4,A              ;"Address"
        LCALL     XmitSync
        MOV       R7,AR4
        LCALL     _XmitAddress
        LCALL     XmitZero
        CLR       A
        MOV       R5,A              ;"Bits"
        MOV       R4,#03H           ;"c"

?C0021:
        CLR       P2_2
        LCALL     Delay25us
        JB        P2_0,?C0022
        MOV       A,R5
        ORL       A,#08H
        MOV       R5,A

?C0022:
        MOV       A,R5
        CLR       C
        RRC       A
        MOV       R5,A
        LCALL     Delay25us
        SETB      P2_2
        LCALL     Delay50us
        DEC       R4
        MOV       A,R4
        JNZ       ?C0021
        MOV       R7,AR5
        MOV       A,R7

?C0023:
        RET
; FUNCTION BIO_READ (END)
```

## MICRO IMPOSTORS

The DS2250 and DS5000 look remarkably like a standard 803 1. This is especially true from the programmer's perspective. To this end, it proves to be a good impostor since a closer examination reveals that it is something else entirely. Figure 1 shows the heart of the DS2250: the DS5000FP. This is the 84-pin circuit that functions as the DS2250's (and DS5000's} processing engine.

Although there are several blocks that may look familiar to users of 8031s, it's obvious that this is no 8031. The designers did a nice job of preserving the basic 803 1 feature set and overlaying this core with a bunch of useful peripherals and enhancements. Many of these new features are accessed through the 803 l's special function registers (SFR). Here, it was just a matter of the designers choosing the most appropriate locations for the new functions, since the existing SFRs have numerous unused bits and bytes.

In my opinion, one of the most useful enhancements is the preservation of all of the I/O ports while still supporting external program and data memory. You can see from the figure that the interface to external memory carried over a dedicated data bus and nonmultiplexed address bus. If you exceed the bounds of the defined partition ranges, address, data, and control information will be emitted over alternate functions of the I/O ports, but normally this isn't the case.

Other useful features include a built-in reset and watchdog as well as a power fail indicator interrupt. The DS5000FP also contains the backup control circuitry for the nonvolatile program and data RAM and the real-time clock. Related memory control functions include the memory allocation and control logic, which defines the total amount of memory and the partition address between program and data memory and the timed access logic that controls access to the processor's critical control and configuration regions.

Since the part is nonvolatile, it should be evident that accidentally reconfiguring the chip could render it inoperative. This is why steps are taken to protect certain sensitive areas. The resident loader ROM provides all the support required for configuring the DS5000FP and for loading and verifying executable program. The loader essentially gives you a live processor right out of the box. That is, before you even transfer a program to it. Aside from communicating with your system, the loader lets you modify memory and I/O ports and dump memory in a variety of formats. Figure 2 reduces the DS5000FP to a box and shows the connections that are made to program/data RAM and to the optional RTC. This is what the DS2250 consists of

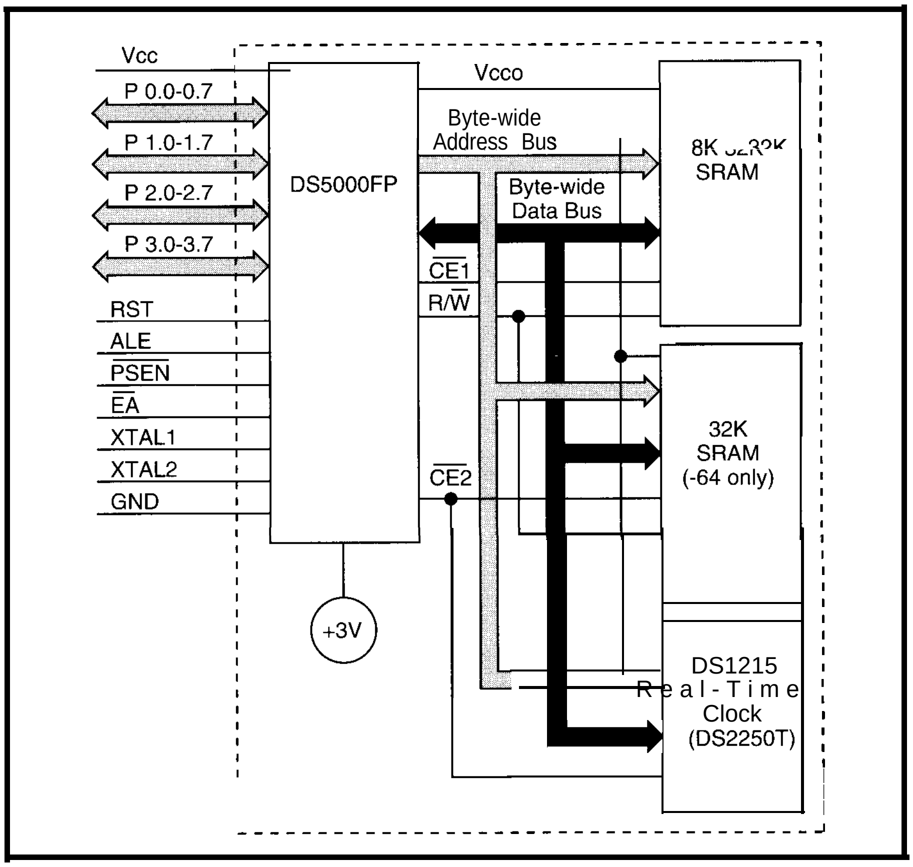


Figure 2—*The DS5000FP uses a resident* loader ROM to provide *all the support* needed *for proper configuration.*

and I, in turn, depict this module in my schematics with nothing more than a box with 40 pins on it.

Now go sprinkle some satellites around your home, office, or factory floor and start collecting data. ▣

*John Dybowski is an engineer involved in the design and manufacture of hardware and software for industrial data collection and communications equipment. He may be reached at john.dybowski@circellar.com.*

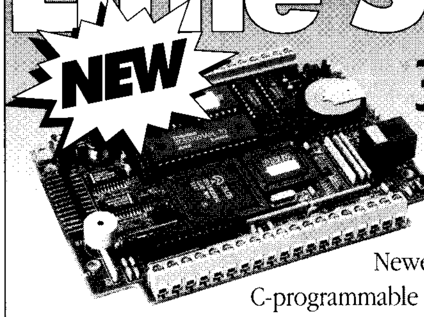
## SOFTWARE

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.

## IRS

422 Very Useful
423 Moderately Useful
424 Not Useful

**Little Star**™



**30 I/O lines**
**Only $195**

Newest in Z-World's line of C-programmable miniature controllers, the Little Star™ has 16 protected digital inputs, 14 high-current driver outputs, RS232/RS485, battery-backed RAM and real-time clock, programmable timers, watchdog, and more. The Little Star is also available with enclosure and LCD/keypad, expansion cards for additional I/O, and optional 18 MHz clock. Our easy-to-use, yet powerful Dynamic C™ development system ($195) integrates an editor, compiler, debugger, and dozens of software libraries all in source code. The Little Star is ideal for OEM control applications, manufacturing automation, test and data acquisition.

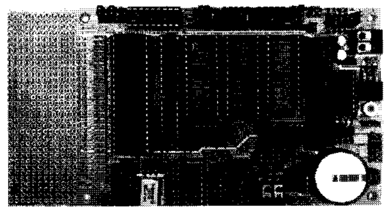
1724 Picasso Ave.
Davis, CA 95616
916.757.3737
916.753.5141 FAX

Z WORLD ENGINEERING

24-Hour AutoFAX 916.753.0618. Call from your FAX. Request catalog #18.

133

**8051 EMBEDDED CONTROLLERS**
***With Lots of Extras!***



We offer a full line of low cost 80C32 embedded controllers and software tools which are ideal for developing products, test fixtures and prototypes.

***Features Include:***
- Low power CMOS design
- Up to 60K of code space and up to 60K of data space
- 5 to 15 volt operation
- Small form factor (3.5" . 6.5") with prototyping area
- System diskette includes application notes
- Start at $100

***Available Options:***
- Multifunction Board adds ND, 24 I/O lines and more!
- BASIC-52 or Monitor/Debugger in EPROM,
- C Compiler $100 or BASIC Compiler for $300

**Iota Systems, Inc.**
POB 8987 . Incline Village, NV 89452
PH: 702-831-6302 . FAX: 702 831-4629

#134

# 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 **Email: sysop@circellar.com**

*This month we're going to traverse the limits from high-voltage concerns down to low-voltage and -current issues. In the first thread, we look at whether the paranoia surrounding potential ESD damage to CMOS devices is still real or is now just imagined. The other thread this month deals with minimum current and voltage that must flow through a set of relay contacts to keep them clean and reliable.*

## ESD-Myth or reality?

**Msg#:21954**
From: MARK SERBU To: ALL USERS

I'm trying to get opinions from anyone who'd like to give one on the subject of ESD. The company I work for is so paranoid about ESD that virtually anything that seems remotely qualified as static sensitive is put in bags, foam, and so forth. To not follow ESD procedure is to get your butt chewed. It's gotten so bad that even a component we stock which is designed to shunt lightning to ground is under the same severe ESD protection as a CD4000 series look-at-me-wrong-and-I'm-history chip!

My own experience with ESD is that it is no longer a factor. I'm using the same CMOS chips (modern HCT types) that I've had and mishandled for years and they still work fine. I've never had a failure of any kind! IS this a big scheme by the ESD paraphernalia manufacturers or is there some truth to it all? I'd love to hear some war stories on this one!

**Msg#:22141**
From: ED **NISLEY** To: MARK SERBU

I'd say it's largely reality with a goodly dose of hype thrown in..

The trouble with the "I've done it hunnerds-a-times and ain't nothin' happened yet" approach to ESD [which I've used myself for quite a while) is that you =can't= tell when you've zapped a chip. The energy required to draw a spark is one or two orders of magnitude higher than that required to damage an IC; worse, the damage may be noncatastrophic so you say, "Well, that chip sure is tough" when you've really shortened its life or increased the leakage or whatever.

My take is that you should exercise reasonable care, use a grounded mat, ground yourself to the mat, and don't do stupid things.. .but you needn't bond yourself to the nearest water pipe and fill the air with ions. Of course, if you've got a manufacturing process that must be certified safe, all bets are off.. .

**Msg#:22345**
From: FRANK **HENRIQUEZ** To: MARK SERBU

I have a good ESD story. About three years ago, we were designing a 128 x 128 infrared camera using a Rockwell NICMOS detector. The 128 x 128 device consisted of a CMOS multiplexer with a layer of HgCdTe (Mercury-Cadmium-Telluride) diodes bonded to the top surface of the mux. This particular device had *no* EDS protection for the CMOS mux, so, not surprisingly, my boss was very paranoid about assembly procedures. We bought a flow bench with a high-voltage grid and ground plane (chicken wire) in front of the air stream that would, in principle, remove all charge from the air. On top of that, we had a grounded work surface on the flow bench and grounded wrist straps.

For some reason, we started killing arrays! We went through the usual list of suspects: detector installed incorrectly, cryogenic failure, you name it, but we were quickly running out of arrays and engineering samples (at $5000–$10,000 each, we didn't have drawers full of them]. Luckily, Rockwell is a fine bunch of people and they wanted to get rid of their old engineering stock for this device, since they had moved on to 256 x 256 devices.

We couldn't figure out why the arrays were dying, so I carefully observed our assembly procedure-the dewar (about 2 feet high) was placed in the flow bench, the air was turned on, wrist straps donned, and the array removed from its box and carefully placed on a grounded surface. Then it was carefully picked up and slowly moved to the socket inside the dewar. It was then that I noticed the high-voltage electrodes behind the flow bench ground grid were about 2" wider than the grid.

As a long shot, I mapped out the voltage potentials (with an ESD meter that I'd borrowed) of different objects in the flow bench. The grounded work surface and array box were at ground, but as I measured the voltage near the top of the dewar, it was well over 40 V! The resulting map showed that the undersized ground grid was letting the HV electrodes charge up everything on the bench! Indeed, after turning off the flow bench, we had no more losses. We also moved on to the larger arrays, which are much easier to work with and do have static protection. We don't use the flow part of the flow bench any more, though.. .

We called the flow bench maker, and their answer was, "Urn, we decided to make the grounding grid a bit smaller.. .I guess it doesn't work, huh?"

**Msg#:22395**
From: DAVE **VANHORN** To: ED NISLEY

I use a technique that I've developed over the years, and confirmed with several chip manufacturers:

1. Wood bench. Not plastic, not metal. It's somewhat conductive, and not prone to generate anything significant.

2. When handling chips, if I need to hand a chip to someone, I drop it into their hand. The chip carries only a very small charge at my potential. Alternatively, for really expensive or sensitive devices, I touch their hand first since that will equalize the charge.

3. When inserting chips into boards, or handling boards in general, I touch an exposed ground-point first.

4. I use a grounded ring around the perimeter on all my boards and attach metal connector parts to that ground ring. It also helps layout tremendously.

The whole idea is to bring everyone to the same potential rather than trying to get to "ground." If we are all at the same potential, then no damage. It's the "bird on a wire" approach. I've seen many designs that were tied to earth ground in an attempt to "fix" an ESD problem that were actually made far worse than before. It's like a boat at anchor with a short chain. The tide comes in, and.. .Glug! I prefer to float the circuit wherever possible and assure that any energy coming in has to see local ground first. I've applied that in several million pieces of shipped product and it works well in real-world conditions.

If I have to tie to earth ground, then I tighten up a bit on protection at the I/O points and try to move sensitive circuitry (memory protection) to the center of the board.

**Msg#:22705**
From: ED NISLEY To: DAVE **VANHORN**

Sounds good to me.. .although you usually have to explain to somebody why you're touching the hand before passing the part to 'em. Such is society.. .

And, besides, wood benches =look= nicer, too. The trick is to justify real wood rather than plastic laminate!

**Msg#:26613**
From: DAVE **VANHORN** To: ED NISLEY

Well, just show them the bill for the complete ESD workstation, then ask if you could just get two file cabinets and a 6' x 3' door. Works every time.

Sometimes they get nervous though with $100k of equipment solely supported by a $20 door. Never had one fail, though!

**Msg#:28206**
From: ED NISLEY To: DAVE **VANHORN**

Hey, and it's probably at the precisely correct ergonomic height, too. Nice work!

**Msg#:25043**
From: PELLERVO KASKINEN To: MARK SERBU

Myth or reality? Actually it can be both, depending on the circumstances. If you operate at relative humidity above 90%, it probably is very close to the "myth." But get down to 15% or below and you could not even blink your eyelids without creating too much static for the chip to survive.

**Msg#:28687**
From: MARK SERBU To: PELLERVO KASKINEN

Pellervo, I guess that here in Tampa we're close to 90% humidity a lot of the year, so that may be why I have such good luck not blowing out parts. I do miss the ability to scrub my shoes on the carpet and give my cat a shock, literally.

## Contact Rating

**Msg#:27007**
From: GEORGE NOVACEK To: ALL USERS

Not long ago there was a discussion on this BBS regarding relay contact rating. The maximum voltage and current in particular. I have a question regarding the opposite end, that is, the minimum allowable current. There are requirements to maintain a certain minimum current through contacts-on a relay or mechanical switch-which depend essentially on what the contact material is.

Since I do aerospace work, I have always played it safe and specified gold-plated contacts. Now one customer is telling me he is not going to use gold-plated contacts and I have to make sure the pull-up resistors put enough current through the contacts for reliable connection. I do not want to go overboard as power dissipation and weight are an important factor. I need data to substantiate whatever current I am going to select. Referring to the old rule of thumb will not fly-literally. Any idea where I can find such information?

# CONNECTIME

**Msg#:27411**
From: CHRIS HAYNES To: GEORGE NOVACEK

I ran into this problem using Potter & Brumfield T92 series relays. The applications engineer there was very helpful and informed me that my relay had a minimum load of 150 mA. Running much below that level caused the contacts to wear quickly. I recommend talking to the manufacturer of the relay.

As a side point I have come across some stuff called "Stabilant R." This stuff is supposed to allow tin-plated contacts to act much like gold contacts. I have a friend who uses it on some terminals in one of his greenhouse projects and he swears by it. However, this stuff isn't cheap; it costs about $50 per ounce.

**Msg#:29648**
From: GEORGE NOVACEK To: CHRIS HAYNES

Thanks for your interest. I know Stabilant, used it, and have nothing good to say about it. In this situation, however, I have no access to the contacts. They are provided by someone else. I only have the contact closure to work with.

**Msg#:28715**
From: PELLERVO KASKINEN To: GEORGE NOVACEK

The low current limit varies widely, as you might guess. If your contacts are built to handle 100 amperes, they are not suitable for the low-level currents. But even the low-level-specific contacts are not infallible. If you have both a low voltage and a low current, the task is more difficult than if one of those two is reasonable.

The basic approaches for low-level contacts in relays rely on bifurcation, which provides redundancy and increased contact force by shaping the two contacts like wedges. Then you have them crossed, to produce a point contact with the associated high local pressure. Penetrates any oxides and so on. Naturally, you want the contact material itself to be as inert as a metal can, without relying on an oxide film for the inertness. Gold is the material of choice in any atmospheric situation.

There also are the sealed (reed capsule and other implementations) relays that have an inert gas filling or a good vacuum. Both these reduce chances of contamination and promote dry circuit (low-current) operability.

Another approach has always been the mercury-wetted contact. While used in some of the best relays for low-level signal switching even today, they are not very popular for the environmental reasons. A high price and position sensitivity in most designs are also impediments.

The latest additions to the bag of tricks are photo switches, such as the PVR series from International Rectifier. Other sources for these photon-actuated FET switches are AT&T and Aromat, plus probably several more. These switches have the benefit of very low thermal EMF, which is beneficial for thermocouple selectors in data logging applications.

Whenever a relay is specified for a "dry circuit" application, you can generally trust that it will work to well below 0.1 mA, but getting the exact figures requires either help from the relay manufacturer or your own trials. It is also a question of how many million operations and what your criteria are for the failure. If the circuit allows a 0.2-V drop in the closed relay contact while carrying 0.1 mA, we are talking about 2-kilohm contact resistance! You normally expect 50 millohms as a reasonable contact resistance, so we are talking about an entirely different expectation here.

As to your change from gold to silver contacts, I would be worried, no matter what power dissipation you design in the pull-up resistors. I cannot see how YOUR customer would accept the change, so you should not either. Aerospace? I hope it is not going to be on any airliner that I'm going to trust for my traveling needs.. .

**Msg#:29650**
From: GEORGE NOVACEK To: PELLERVO KASKINEN

Thanks for your reply. I have been in the business longer than I care to remember and am well aware of methods to do it right. Unfortunately, this is a situation where I am designing a flight control box, with the aircraft interfaces outside my design authority. The ones in question are discrete signals, such as weight on wheels, which were defined to me as either open or closed contacts. Only now, after the fact, the airframe manufacturer came to us and said, "By the way, the contacts will not be gold plated but rather silver plated, rated for 25 A. We are concerned that your pull-up current is insufficient and expect you to either prove to us otherwise or increase the current accordingly. "

Fortunately, it is not a safety issue. As you may know, in my business the functionality is a given. Achieving it represents hardly more than 20% of the design effort. We spend more time designing for failure because everything will eventually fail and we must ensure it fails in a safe, predictable manner. Because of the system redundancies and BIT (built-in test], I will know when a contact has failed. Also, because of the noise margin built into the input circuits (I need to operate in E-fields of 400 V/m and take Level 4 lightning hit), I can tolerate up to 3k contact resistance. The issue is primarily maintainability. Once there is high incidence of contact failures (detected and indicated by my equipment), I will get the blame. Always blame the messenger! :-(

I am trying to alleviate this very real potential problem by convincing my customer it would be cheaper to spend a few hundred bucks extra building a $15 million airplane than blowing a lot more later in a retrofit program. It is difficult to convince someone to spend money by using arguments such as experience, rule of thumb, and so forth. For an engineer (as well as a bean counter), there is nothing more convincing than a good set of numbers. That is what I was looking for when posting my message.

In the meantime, I talked to several relay manufacturers in hopes of obtaining solid data. Basically, they all tell me that anything less than 10% of the rated current should be considered low load and viewed with suspicion. Based on that, I should have at least 2.5-A pull-up current. That, of course is ludicrous, since the entire controller is allowed to consume no more than 35 W. From this perspective it also means it makes no difference whether I stay with 1.2-mA pull-ups or increase them to 5 mA (absolute maximum for the allowed power). Either way, I am still a long way from 2.5 A. The 3k contact resistance tolerance seems to be a big plus. Manufacturers of high-quality relays, such as Leach, can implement a screening program which will test the 25-A rated devices at their intended load. But even then, anything other than gold plating raises eyebrows.

My design has to pass the salt fog, sand, and dust test. If you have ever seen the inside of a dual-cavity box with only a single 0.020" hole for pressure relief after such tests, you would be wondering why would anyone even consider using nonhermetically sealed, nongold-plated relays in such an environment or, conversely, if such environment is not real, why would he want any piece of equipment meet such environmental conditions. It ain't cheap to achieve.

It looks like there is no solid data I can take and beat over my customer's buyer's head. Probably my only course of action is to write a memo, strongly recommend they use gold-plated, hermetically sealed, low-load screened contacts, send it out, and when eventually the sh** hits the fan, wave it around and say I told you so.

**Msg#:30843**

From: PELLERVO KASKINEN To: GEORGE NOVACEK

Pretty much what I figured your situation might be. Sounds to me like you are in a no-win position. Twenty-five-amp contacts! Those relays have almost dime-size contacts and I am every bit as scared as I was last time about the choice. Why in all the world would the spec need such monsters? Is it just another case of standardizing on a single type of something for minimizing the inventory requirements?

**Msg#:31532**

From: GEORGE NOVACEK To: PELLERVO KASKINEN

That's what it primarily is. Minimizing inventory and getting a volume discount from the vendor. Also an attempt to minimize certification costs by marrying old technology with new. In the aircraft industry, there is what is referred to as grandfathering. Essentially, you use as many old parts and systems as you can and claim they are being grandfathered. If accepted, it reduces the time, effort, and cost to certify the particular part or system tremendously. Boeing 737 has been grandfathered so many times I lose count—although, I need to add, there is nothing wrong with it, and Boeing, in particular, never compromises on quality and safety. What usually puts a stop to endless grandfathering is a new requirement. Recently, for example, new regulations increased HIRF/EMI requirements and included lightning strike susceptibility to such proportion, very little electronic equipment can avoid redesign.

The aerospace industry is extremely conservative-it must ensure personal safety. Most of the "cutting edge" technology was done on military projects. There, funding used to be plentiful and a loss of an aircraft was to some degree acceptable. This is not the case in the commercial part of the industry. Basically, equipment that has been flying for a couple of years is specified even for the future projects and something new is procured only if the old and tried stuff does not exist. Fortunately, in my area everything must be designed from scratch. But I caused quite a commotion in a boardroom of one company when I proposed to use fuzzy logic. I am using it anyway, an algorithm is an algorithm. I just don't use the silly name any more.

Back to the relays. It is not a safety issue and, my tests show, it will work for quite some time. It is just silly and I don't like it. I in fact wish I could show it compromised safety. Because then it would be changed even before I finished writing this reply.

**Msg#:33802**

From: PELLERVO KASKINEN To: GEORGE NOVACEK

I have been thinking this issue for the past week, trying to come up with a constructive suggestion. I came up with something, although I cannot claim right away that it is terribly constructive.

A little background: I once subscribed to a Swedish radio and electronics magazine. They had in every issue some outlandish trick question that the readers were invited to answer, sincerely or not so sincerely, depending on each person. Then the best of the answers were printed in a later issue. Now, one of the questions was about a black box that from a DC supply consumed 1 A of current while the terminal voltage was 1 V. Yet, a watt meter also attached showed 0 W. All instruments were perfectly functional and accurate. What was there in the black box? The answer was a chopper contact that produced a 50% duty cycle, a short and an open. Oh, I failed to mention the

# CONNECTIME

internal resistance of the DC power supply was specified as well.

What all this means is you could have plenty of current and very low power dissipation if you choose to chop the sense current. You would provide in essence an inductive path whenever the external contact is made and no path when it is open, but you would have the chopper and other energy reuse facilities built right into your portion of the circuit.

Of course, you could just try lowering the duty cycle of ordinary sampling pulses, never resort to the inductive circuits. Just repeat a 0.2-ms sampling pulse of 1 ampere or so from 5 V a few times per second. Charge up a capacitor to provide the jolts..

Well, I do not know what consequences the pulsing would have to your other parts and the noise issues and so on, but at least I came up with something to further the thinking processes of the creative minds among the subscribers to this board, didn't I?

**Msg#:33937**
From: GEORGE NOVACEK To: PELLERVO **KASKINEN**

An interesting idea. I did tell the customer to do his part of the job right and give me proper contacts. However, it is something to keep in mind for the future.

There are some minor problems, such as the fact that we usually work with 16 to 32 discretes. To keep the cost down, I could always multiplex them. A typical system has a dozen analog inputs which are multiplexed anyway, so what's a few more lines? The test current would be injected through the multiplexer.

Another problem, of course, is the EM1 spectrum. A square wave is a big no-no. Therefore, most data communication systems use trapezoid, which is fairly accurately defined. This goes hand in hand with the redundancy requirements. In a typical fail-safe system, I work with dual redundancy (triple for fail operative). So, I have two inputs with identical data and two microprocessors to process them. Input 1 and 2 must tally on µP1 as well as µP2 and then µP1 and µP2 compare their results and they must tally as well. In addition, to make sure I am not reacting to transients, I must have a failure condition to persist for three to four counts. In total, however, I must be able to declare fault and take action in 80 to 100 milliseconds.

Simple mathematics will show you that the time slice for data acquisition gets shorter and shorter. Some time can be saved by synchronizing both microprocessors, but it is tricky. You must make sure there is no single point failure in hardware, nor common mode failure in software (which means the two programs must be different and sometimes even the micros are different; some people go as far as using one from Motorola, the other from Intel). All that cuts into the time and once we sta[illegible] ,utting out voltage or current pulses, the shorter they ar[illegible] the higher their spectrum. Both radiated and conducted er[illegible]sions can become a nightmare real fast.

**Msg#:30941**
From: BEN MEHLMAN To: [illegible]ORGE NOVACEK

Sounds like officialdo[illegible] at its best. Certainly you shouldn't have to be deali[illegible] with a 25-A relay contact for a control input, and that is [illegible] root of the problem. There are power relays out there wit[illegible] separate sets of control contacts....

Anyway, assuming yo[illegible] an't get the problem fixed at the source, here's a suggest[illegible]n: Use an arrangement with a capacitor to provide a high [illegible]urrent briefly during contact closure. Once the contact i[illegible]established your lower "sealing" current might be adeq[illegible]ate. Mind you I am not a relay expert, but it seems like it [illegible]ould increase reliability.. .

**Msg#:31531**
From: GEORGE NOVACEK T[illegible]: BEN MEHLMAN

An interesting idea, al[illegible]lough something like that already exists because of t[illegible] debouncing caps on the input. I'll look into it. Thanks.

*We invite you call the Circ[illegible]rit Cellar BBS and exchange messages and files with ot[illegible]er Circuit Cellar readers. It is available 24 hours a day a[illegible]l may be reached at (203) 871-1988. Set your modem for [illegible] data bits, 1 stop bit, no parity, and 300, 1200, 2400, 9600, [illegible]r 14.4k bps. For information on obtaining article software [illegible]hrough the Internet, send Email to info@circellar.com.*

## ARTICLE SOFTWARE

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

To order Software on [illegible]isk, send check or money order to: The Computer Applicat[illegible]ns Journal, Software On Disk, P.O. Box 772, Vernon, CT 0[illegible]066, or use your VISA or Mastercard and call (203) 8[illegible]5-2199. Be sure to specify the issue number of each disk [illegible]u order. Please add $3 for shipping outside the U.S.

## IRS

425 Very Useful 426 Mo[illegible]erately Useful 427 Not Useful

# STEVE'S OWN INK

## One Man's Junk...

I'm sure you've heard the expression that one man's junk is another man's treasure. I wouldn't have believed this if I hadn't experienced it so often. Apparently it has gotten around town that I have some of the best junk. Whenever a carpenter or handyman does work around our property they go home with a trunkful. Last summer I took down a 2-car garage (replaced it with a larger garage and workshop) and, believe it or not, none of the contents ever made it to the curbside pickup. Someone even dismantled the garage and took that away!

We engineers shouldn't laugh too loudly about collecting things. Somewhere you probably still have your first computer (or first couple), a bunch of hardwired interface kludges, and piles of old technical books. Why do I still keep those college statics and dynamics texts?

We collect some objects because they have special meaning or remind us of specific turning points in life. I have eight original Tinney paintings hanging on the walls of our living room. I purchased them from Robert because I wanted a distinctive and unique reminder of past associations and achievements, When I look at my painting of the robot hand breaking out of the egg (Technological Breakthrough), it stimulates thoughts of recent evolution.

I'm embarrassed to admit that apparently many of my project presentations have had similar effects. Often I am told that a particular project caused someone to change a career path, win a science fair scholarship, save their senior project, start their own business, or just plain incite new awareness. One guy wrote and told me (back in '78) that my using an EPROM to replace sequential logic saved his job.

Well, embarrassed or not, I've decided to clean house. The Circuit Cellar is getting a little tight from old project boxes and folders going back 15 years. If you are among those who had thought about asking for a unique reminder of that critical juncture in your past, now is the time. If you want some Ciarcia original junk-for mere shipping costs-perhaps we can resolve mutual goals.

First of all, there are over 100 projects going back to '77. If you have a particular project in mind, let me know and list some choices. Also tell me if, in lieu of those, anything will do. The early years are primarily folders containing the raw manuscript and perhaps the author's proof. Most of these early manuscripts are handwritten! The more recent projects might still have chip samples or hand-wired boards.

I offer this stuff as unique junk-first come first served. I don't guarantee the complete contents of anything, that anything included works, or that it's anything more than a unique piece of Ciarcia junk. I'll send it to you and you reimburse me for the shipping after you receive it. Frame it and forget it.

I've kept this stuff this long because it meant something to me. I relish the thought of it finding a home with someone it has positively influenced. To participate, write me here % Ciarcia's Project Boxes.

Steve