CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

EMBEDDED INTERFACING

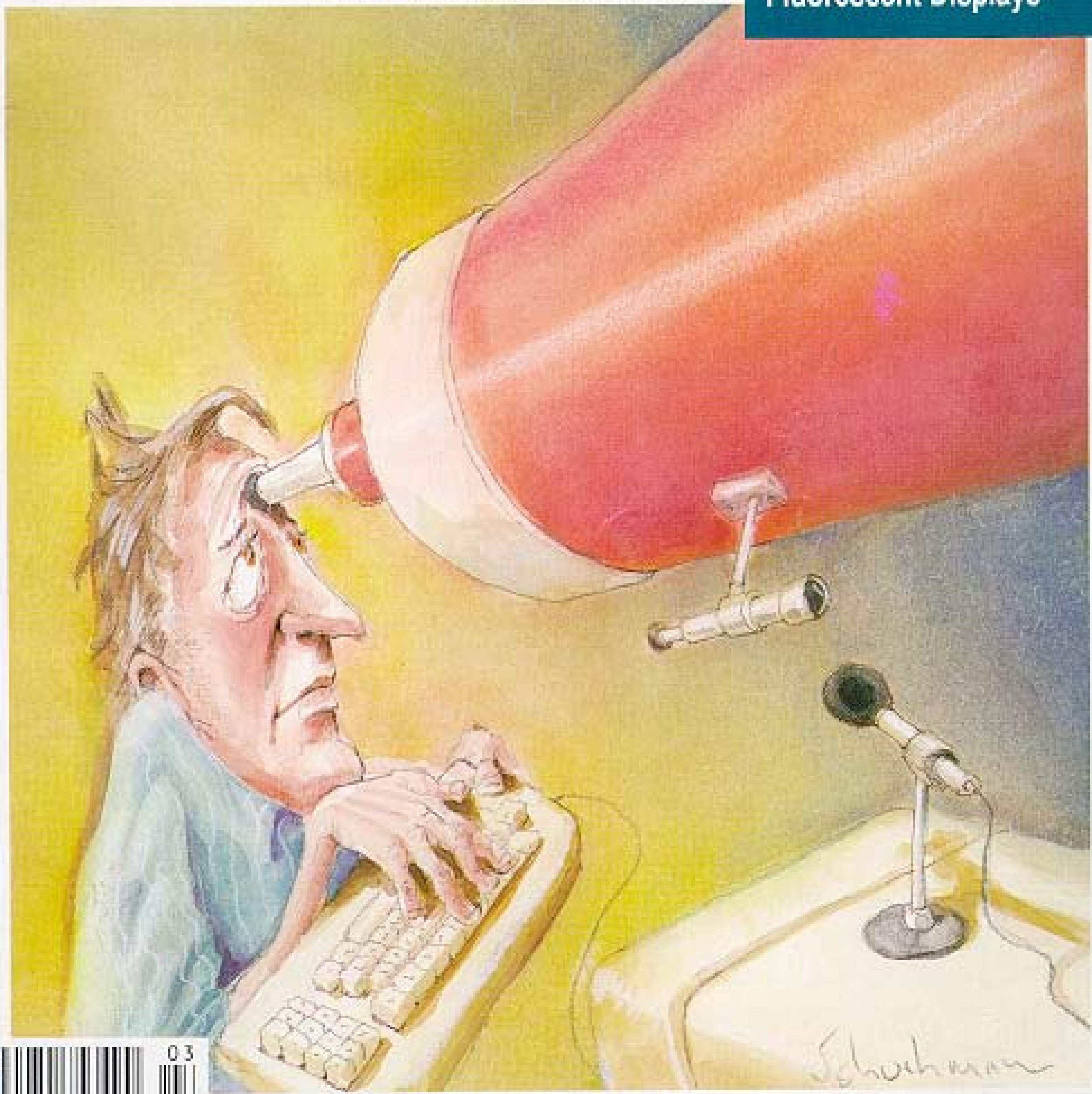
Single-chip Video Wind Gauge

Voice-controlled Telescope Positioning

Dim Lights Under Computer Control

Using Vacuum Fluorescent Displays

March 1993 — Issue #32



03

0 74470 75349 0

$3.95 U.S.
$4.95 Canada

EDITOR'S INK

Getting the Hang of It

Things are finally starting to fall into place with this monthly schedule. We all know what stage each issue is at (should be at?) simply by what time of the month it is. We used to be able to finish putting an issue together before having to worry about the next one. Now, as I write this, we're putting the finishing touches on this, the March issue; we've finished the editing and rough layouts for the April issue; we have all the feature articles in-house for the May issue; and I'm making sure articles for the June issue are being written. I'm also continuously looking six months to a year down the line to be sure we have plenty of editorial in the works.

We start this issue with the first-place winner of the latest Circuit Cellar Design Contest: the video wind gauge. Using little more than a single PIC processor, Philip Pilgrim's project measures the wind speed and direction using four propellers and generates a composite video signal to display the result. A single coax cable runs power up to the propeller/processor unit and runs the video signal back down to the house.

Next, we have the first of a two-part article on how to dim incandescent lights under computer control. While the focus of the article is on stage lights, the ideas can be applied just as easily to household lighting or even a Las Vegas marquee.

The voice-controlled telescope project is another winner from a past design contest and is just as elegant in its simplicity as the wind gauge. You'll need your own telescope, though.

The final feature describes a clever technique for adding interrupt support to systems using 8255 PPIs wired only for polled operation. By applying this design trick, you can set your system up so the circuit generates an interrupt any time an input bit changes state.

In our columns, Ed continues with his embedded '386SX system by showing how to get a C program to run without DOS; Jeff uses the Maxim MAX7219 seven-segment LED display driver chip in a sample circuit; Tom looks at a 16-bit processor that makes a worthy candidate as the core of embedded systems that need more power than typical 8-bit microcontrollers can offer; John tosses aside the traditional LED and LCD displays in favor of the less-often-used vacuum fluorescent display; and Russ describes patents that cover PAL design tools, dual cooling fans, text compression and decompression, and a microprocessor-based electronic book.

Ken

CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

[redacted]

FOUNDER/EDITORIAL DIRECTOR
Steve Ciarcia

EDITOR-IN-CHIEF
Ken Davidson

TECHNICAL EDITOR
Robert Rojas

ENGINEERING STAFF
Jeff Bachiochi & Ed Nisley

WEST COAST EDITOR
Tom Cantrell

CONTRIBUTING EDITORS
John Dybowski & Russ Reiss

NEW PRODUCTS EDITOR
Harv Weiner

ART DIRECTOR
Lisa Ferry

GRAPHIC ARTIST
Joseph Quinlan

CONTRIBUTORS:
Jon Elson
Tim McDonough
Frank Kuechmann
Pellervo Kaskinen

PUBLISHER
Daniel Rodrigues

PUBLISHER'S ASSISTANT
Susan McGill

CIRCULATION COORDINATOR
Rose Mansella

CIRCULATION ASSISTANT
Barbara Maleski

CIRCULATION CONSULTANT
Gregory Spitzfaden

BUSINESS MANAGER
Jeannette Walters

ADVERTISING COORDINATOR
Dan Gorsky

CIRCUIT CELLAR INK, THE COMPUTER APPLICATIONS JOURNAL (ISSN 0896-8985) is published monthly by Circuit Cellar Incorporated, 4 Park Street, Suite 20, Vernon, CT 06066 (203) 875-2751. Second class postage paid at Vernon, CT and additional offices. One-year (12 issues) subscription rate U.S.A. and possessions $21.95, Canada/Mexico $31.95, all other countries $49.95 All subscription orders payable in US funds only, via international postal money order or check drawn on U.S. bank. Direct subscription orders and subscription related questions to The Computer Applications Journal Subscriptions, P.O. Box 3050, Southeastern, PA 19396 or call (215) 630.1914. POSTMASTER, Please send address changes to The Computer Applications Journal, Circulation Dept., P.O. Box 3050, Southeastern, PA 19396.

Cover Illustration by Bob Schuchman

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

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

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

INSIDE ISSUE 32

READER'S INK

AN UNHAPPY CUSTOMER

I was sorry to see you give Dallas Semiconductor such a glowing response in customer service in your February 1993 "Steve's Own INK." Perhaps our coal mining environment is just too severe for their parts, or perhaps they are just junk.

This is a chronology of my company's experience with Dallas parts:

The DS1210 memory backup part. Never did get the reliability out of the thing that we needed. It would back up memory for hours or days, then the memory forgot everything. So we didn't design it in, and used the diode/ transistor approach in the design. I've seen the 1210 in several of your designs; what kind of experiences have you had with it!

Next we designed in some of the very first DS1201 "tag" parts (1024 bits of NVRAM). After much hair pulling, we found that they were very susceptible to ringing on the clock line. The parts didn't just not work, they died. On a call to customer support, we were told "Yes, we have seen that before. You need to put a low-pass filter on the clock line." "Why wasn't that in the data sheet?" Never did get a good answer to that. If you dig through their application notes, you might find that piece of information in there now.

Next, the DS1216 SmartWatch. Always one to be on the bleeding edge, I got one of the first ones. I had a simple application: using it on one of your SB180 boards. When I got my sample, I got a data sheet that I could just about read because of the poor photocopying. A call to customer support went, "Do you have any sample code for dealing with this thing?" "Yes, we do. We will send it right out." Why didn't they send it with the parts in the first place? When I did get it, it was also almost illegible, and it didn't even work. I spent many long mornings (I'm a morning person rather than a night person; I do my best work between about 4 A.M. and 11 A.M.), as Ken may remember, working on my SW TIME program.

In all of my testing of the DS1216, I was never real happy with its accuracy, but I went ahead and designed it into one of my coal mine controls. It wasn't long before I realized that the accuracy had gone completely to pot. A call to customer service got, "Are you driving the 1216 with full CMOS levels?" "Yes, of course, we need the noise margin in our environment. Why?" "Well, when you drive the 1216 with CMOS levels, just the slightest bit of overshoot makes the oscillator stop running until the voltage goes back down." "Why wasn't that in the data sheet?" Never did get a good answer to that one. I don't know if that is in any of their current information or not; I don't have a data book with me here.

After these experiences, we started to avoid Dallas parts like the plague. While I did get answers to my questions from customer support, they were questions that I should not have had to ask. The information should have been in the data sheets.

But, no. We're not done with Dallas yet.

Remember those DS1201 Tags I mentioned? Well after we solved the ringing problem with a low-pass filter, we went ahead and designed them into our equipment.

You can get a cap for the 1201 called a Key Chain-a small plastic cap that you glue on to the 1201. It gives you something to grab hold of to extract the thing.

Early last year, production brought me a bag of a thousand of these things. "None of these fit!" "What do you mean they don't fit?" "We can't get then into the slot in the unit." "Let me try one." Sure enough, these caps had gotten several thousandths of inch thicker all the way around (the wall thicknesses had increased).

A call to customer support got, "Oh, yes. We changed the way we make the things so that the walls aren't so thin. We kept having trouble with them splitting during manufacturing." "They don't fit in my equipment anymore!" "How many do you have?" "A thousand." "Tell you what we'll do. We'll mill a thousand of then down to their old size for you and send them out, but after this you'll have to mill them down yourself." Gee thanks.

Last month the purchasing department came to me and said, "Do you know of any replacements for the DS 1201? Dallas now says they are 'proprietary."' "How can they be proprietary? We've been buying them for years." "That's what the distributor told us the factory told them." Purchasing solved this problem on their own. They are now buying the 1201s at three times the price that we were paying for them when they were not "proprietary" two months ago.

So, let's hear it for Dallas Semi!

Bob Paddock
Masury, Ohio

Sorry, ***Bob, but all your problems with Dallas sound like my experiences with National Semiconductor. I guess*** we ***all have our problem companies. Other than it taking a long time to figure out the Dallas clock stuff,*** we've ***always had good luck with Dallas parts.-Steve***

READER SUGGESTIONS

All of your articles involve serial processors, which are cheap, versatile, and well-suited to most applications. However, have you considered an article on parallel processors (e.g., T800 from INMOS)? I admit they are not cheap and are reasonably difficult to program, but they do an excellent job where there is substantial data to be processed.

Jeff Hutchinson
Burren Junction, Australia

I have really enjoyed the *Computer Applications Journal.* How about some info on TBCs, procamps, switchers, black-burst generators, SFX, and video switchers? Neat stuff.

Dave Freeman
Sacramento, Calif.

I am an aspiring artist and am very interested in making my art interactive. I have many ideas for incorporating the use of microcontrollers into my art such as for motor control, light control, and other bells and whistles. How about an article on combining microcontrollers with art?

Ronald Brown
Upper Black Eddy, Penn.

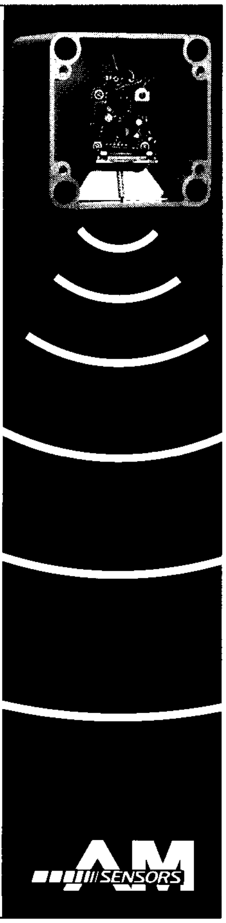
We at the Computer Applications Journal would like to publish articles on all these subjects. Unfortunately, the solution involves a combination of expertise and available space. While the ideas for articles come from both readers and staff, most of the editorial is written by experts and engineers. Our editorial consists of both independent submissions and solicitations.

Because there are only so many pages in each issue, however, we can't cover all subjects. Of course, if you are or you know of an author with such expertise, please let us know and we can put a hot subject on the front burner faster.

We Want to Hear from You

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

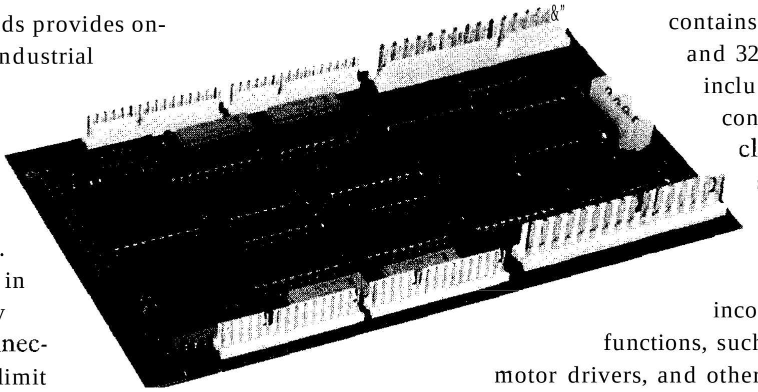
The Computer Applications Journal
letters to the Editor
4 Park Street
Vernon. CT 06066



NEW PRODUCT NEWS

Edited by Harv Weiner

COMPUTER I/O NEEDS NO BACKPLANE



A series of IDE I/O boards provides on-board control circuitry for industrial input and output without the need for a backplane. The Series 2100 **I/O** Expander boards from DDI Inc. communicate through the IDE hard-drive interface. The boards are ideal for use in output applications for relay drivers or sensing input connections for push buttons and limit switches.

Series 2 100 boards communicate through an interface already standard on most single-board computers and do not require separate plug-in I/O boards. This aspect reduces the space requirements of embedded control circuitry by eliminating the need for bus backplane cards. A single IDE I/O expander board replaces two to six PC or STD I/O bus boards at one-fifth the cost.

The I/O expander board contains 16 solenoid drivers and 32 inputs. Typical uses include sensing and control of contact closures, limit switches, and solid-state sensors. Other models are expected to incorporate additional functions, such as isolated I/O bits, motor drivers, and other I/O combinations.

All Series 2100 boards require only ribbon cable connection hardware and can be powered from the single-board computer's 5- and 12-volt power supply.

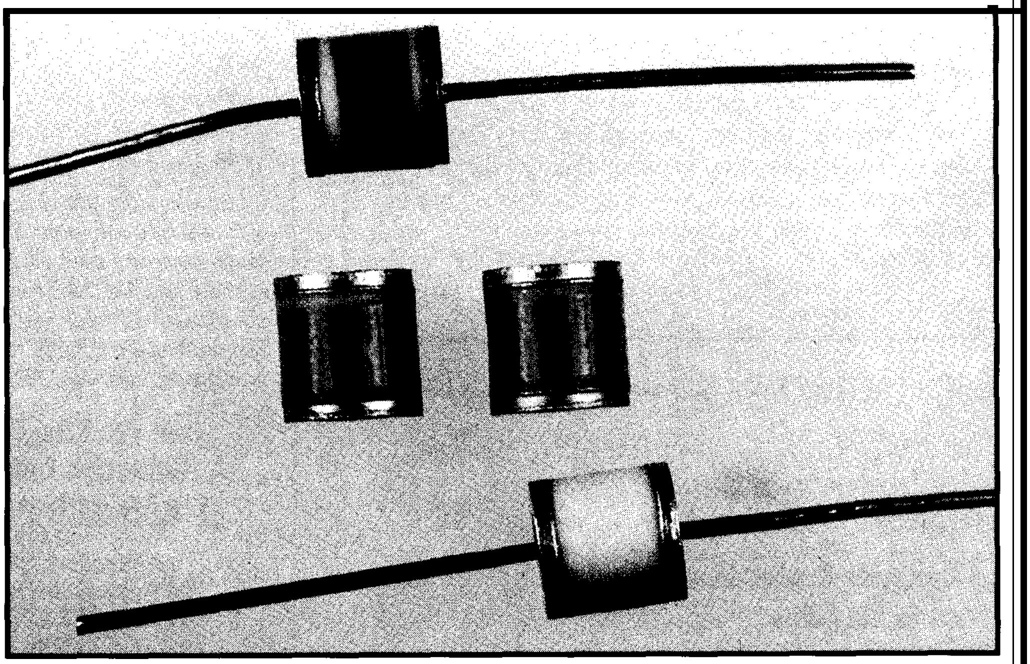
DDI, Inc.
5 Dodge St., Ste. 310 . Beverly, MA 01915
(508) 927-7976 . Fax: (508) 921-6388

#500

HEAVY DUTY SURGE ARRESTORS

Lumex Opto/Components Inc. announces a new family of heavy-duty ceramic overvoltage gas tube arrestors, capable of supporting surges of 40 kA (8/20 us) waveform. This level is twice that of current protection from lightning, power failures, switching power loads, and other natural or man-made malfunctions of other currently available arrestors.

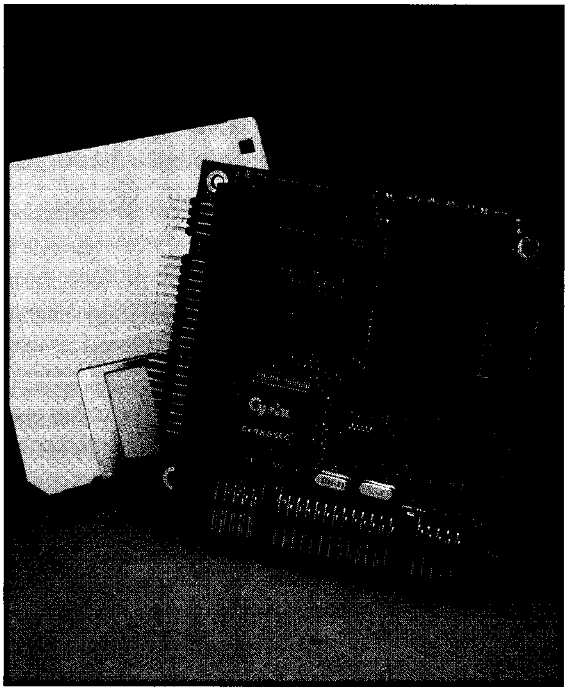
The increasing use of electronically controlled devices in medical, industrial, and security control equipment, as well as outdoor installations, such as broadcasting and receiving antennas, create a need for overvoltage protection. Models are available in firing voltages of 90, 230, 350, and 470 VDC, with a choice of button types or with welded leads for printed circuit board installation.



Lumex **Opto/Components,** Inc.
292 E. **Hellen** Rd.
Palatine, IL 60067
(708) 359-2790
Fax: (708) **359-8904**

#501

NEW PRODUCT NEWS



486-BASED COMPUTER MODULE

Ampro Computers Inc. introduces a 486-based computer module that offers complete PC/AT compatibility in under 14 square inches. The **CoreModule/486** meets the new 3.6" x 3.8" PC/104 form factor and includes the equivalent functions of a PC/AT motherboard plus several additional expansion cards. It has a 25-MHz CX486SLC microprocessor, 4 MB of DRAM, a serial/parallel controller, a keyboard port, and speaker interfaces. An onboard, bootable, l-MB solid-state disk, watchdog timer; and power monitor functions ensure high system integrity and reliability.

You can use the CoreModule/486 as a component-like module, plugged into a custom circuit card, or combined with Ampro expansion products to form a compact, highly integrated control subsystem. Stack multiple modules together, or mount them separately on a custom circuit card.

Ampro has included enhanced BIOS services and functions that meet the requirements of embedded microcomputer system applications. These improvements consist of solid-state disk support that substitutes EPROM or NVRAM devices for normal floppy drives, SCSI services that support DOS-based system operation, and serial console operation. Other features include a serial loader to download executable code from a remote device prior to system boot, and EEPROM access functions for high-level access to the 5 12-bit nonvolatile OEM data area in the configuration EEPROM.

The CoreModule/486 has provisions for an 80387 math coprocessor and operates with approximately 4 watts of +5-volt-only power.

The CoreModule/486 sells for $995 in quantity.

Ampro Computers, Inc.
990 **Almanor** Ave. • Sunnyvale, CA 94086
(408) 522-2100 • Fax: (408) 720-1305

#502

CUSTOMIZED ADAPTER CATALOG

Emulation Technology announces the availability of its 1993 VSLI and SMT **Interconnect Solutions** catalog. This new 160-page catalog includes over 3500 time-saving adapters and accessories used with such development systems as emulators, logic analyzers, and memory and logic programmers.

Over 500 new adapters have been added, including test clips, programming, and prototyping adapters for over 63 different PQFP (Plastic Quad Flat Pack) packages.

The catalog features a Microprocessor Tool Selection Guide, which groups adapters and accessories by microprocessor and manufacturer. It provides detailed listings of all the adapter support for ten different microprocessors, including debugging tools, insertion and extraction tools, and socket converters.

The new catalog is free.

1993 VLSI & SMT Interconnect Solutions Catalog
ET
One-Stop Shopping for over 3,500 Time Saving Solutions for Product Development
Emulation Technology, Inc.

Emulation Technology, Inc. ● **2344** Walsh Ave., Bldg. F • Santa Clara, CA 95051 • (408) 982-0660 • Fax: (408) 748-2675 **#503**

NEW PRODUCT NEWS

CTIVE FILTER DESIGN SOFTWARE

FILEDES Ver 3.1 is an advanced active filter design rogram from Consulting and Training Services (CTS) hat gives a designer the ability to design, optimize, and nalyze active filters interactively. FILEDES operates in he PC-DOS environment and uses well-documented lassic filter functions to approximate the filter requirenents as specified by the user. The program uses a nenu-driven input format to promote efficiency and ffers default values, where appropriate, to aid further in he design.

Some of its features include a unique gain sensitivity roduct minimization algorithm to account for both assive and active sensitivities, interactive design, ten ctive filter topologies, intuitive menu-driven interface, creen/printer plots and hard-copy report summaries. It reates an ASCII file for each active filter design for omplete documentation.

FILDES can implement low-pass, high-pass, bandass, band-reject, and user-defined general-purpose filter types. The filter types may then be synthesized with Butterworth, Chebyshev, Elliptic (Cauer), Bessel (linear phase), or general-purpose Biquad functions. The Transfer Function or Pole/Zero editor provide additional design capabilities.

FILEDES implements the active RC sections with a wide variety of well-defined circuit topologies. Unlike "cookbook" filter designs, FILEDES allows the circuit designer to optimize resistor and capacitor values while selecting from low-Q, med-Q, and high-Q topologies to minimize the Gain Sensitivity Product Function.

FILEDES sells for $79.95 and includes free upgrades for 90 days, phone support, and a quarterly newsletter filled with tutorials and tips on active filter design.

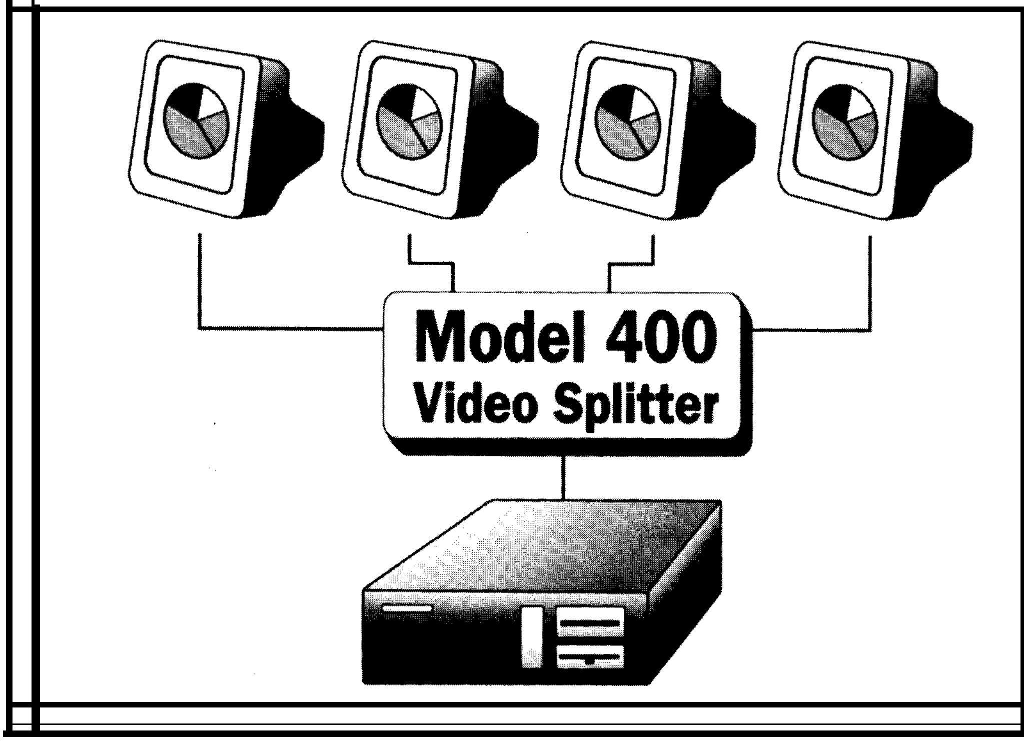
Consulting and Training Services
P.O. Box 223 . Sharpes, FL 32959
(407) 633-9868

#504

'IDEO SPLITTER

The Model 400 Quad 'ideo Splitter from H&R 'echnology provides the ieans for displaying a ersonal computer's ideo output on four or iore monitors. The device buffers and amplifies the video signal for each of the four outputs with a bandwidth greater than 75 MHz. The result is a crisp, undistorted reproduction of the video image on all monitors, even at distances up to 250 feet; for greater lengths or additional monitors, combine two or more splitters.

The Model 400 is compatible with all VGA, SuperVGA, and PS/2 monitors. The unit requires 110 or 220 VAC for operation and features special circuitry to power up automatically when the computer is turned on. The Splitter measures 7.25" x 2.75" x 1.25" and includes the cable for connecting to the computer.



Typical uses for the Quad Video Splitter include conferences, demonstrations, classrooms, industrial control, and monitoring systems.

The Model 400 sells for $325.

H&R Technology
1506 Brookhollow Dr.,
Ste. 106
Santa Ana, CA 92705
(714) 641-6607
Fax: (714) 966-I 770

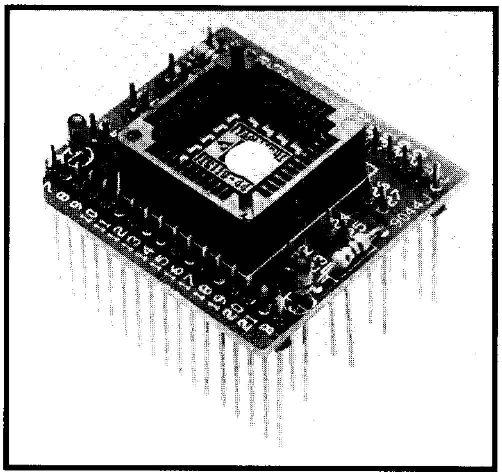
#505

NEW PRODUCT NEWS

LEADLESS CHIP PROTOTYPING ADAPTER

Antona Corporation has provided the circuit designer with a method to wire-wrap prototypes using plastic leaded chip carrier (PLCC), leadless chip carrier (LCC), or pin grid array (PGA)-type components. The model ANC-9044 is a 44-pin adapter occupying just under three square inches of board space and provides test points for each of the pins. This feature allows the designer to attach oscilloscope or logic analyzer probes to the component side of the prototype card. The adapter pins are on 0.3" centered rows so the adapter is usable on a wide variety of prototype boards.

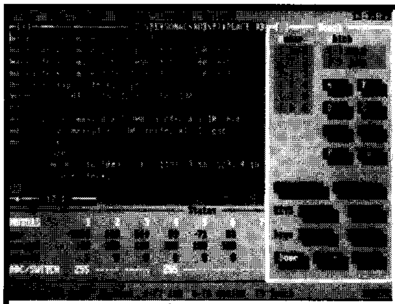
Two LED status circuits on the chip adapter provide a visual indication for user-selected signals. This feature is unique to adapters manufactured by Antona. Additionally, to aid the designer in using the adapter, the user's manual includes an adhesive-backed pin-numbering sheet to guide the wire-wrapping of prototypes and a template of the adapter, which the designer can use as a signal-to-pin designation map. The adapter is available with either three-level wire-wrap pins or gold machine pins. Socket versions are also available to accommodate 20-, 28-, 32-, 52-, 68-, and 84-pin components.

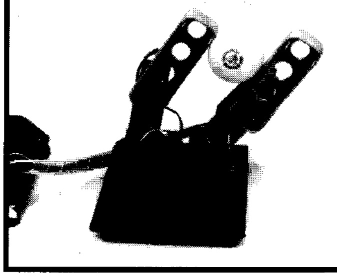


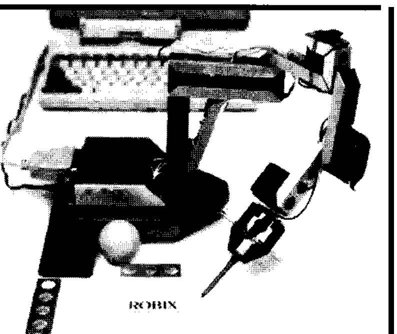
The ANC-9044 sells for $36.

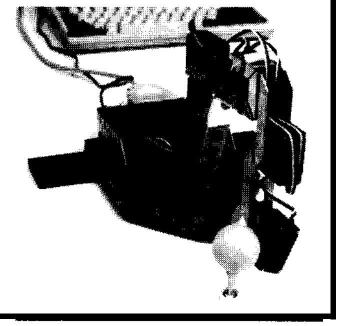
Antona Corporation
1643% **Westwood** Blvd.
W. Los Angeles, CA 90024
(310) 473-8995
Fax: (310) **473-7112**

#506







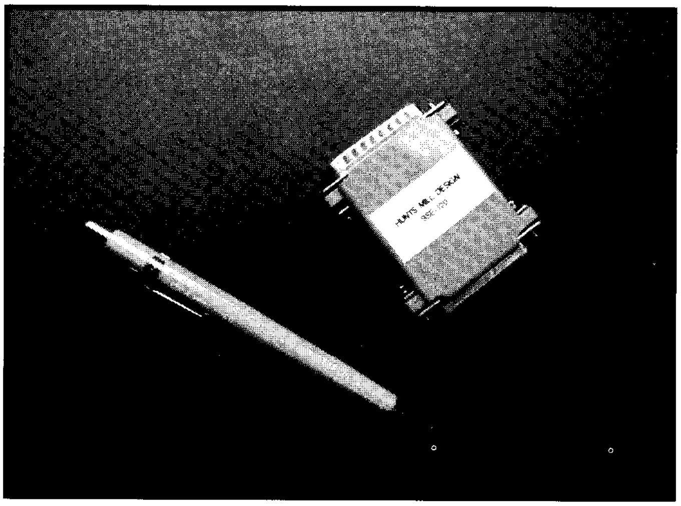


NEW PRODUCT NEWS

SMART QUADRATURE ENCODER

The SSE-100 series of smart quadrature decoders from Hunts Mill Design converts quadrature signals to serial data. Typical applications include robotics, optics, medical instruments, and process automation. The SSE-100 features RS-232 ASCII communication through an operating temperature range from 0°C to 70°C and draws only 100 mA at 5 VDC. Optional features include RS-422 communications port, 4–20-mA output and wide input supply voltage capability.

When coupled with a shaft encoder, the SSE-100's smart interface can remotely sense shaft position, speed, and direction. The smart serial interface allows use with many industry standard quadrature output devices and PLCs, PCs, or custom-designed computers.



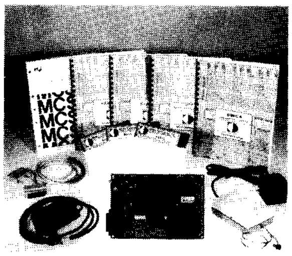
The SSE-100 sells for $80.

Hunts Mill Design
112 Hunts Mill Rd.
Clinton, NJ 08809
(908) 283-3340

#507

The $595 Solution
to 8057 System Development

PDK51



The PDK51 is a fully integrated hardware, firmware, and software system designed to help you develop your products quickly and cost effectively.

All you need to use the PDK51 is an IBM-PC/XT/AT or compatible. We supply the rest.

- *SIBEC-II microcontroller board (8052AH-BASIC CPU) with 16K RAM*
- *BASIC interpreter source on disk*
- *SIBEC-II hardware manual with schematic*
- *Monitor/debugger ROM plus manual*
- *Monitor/debugger hardcopy source listing*
- *UL listed 5V power supply*
- *Programming power supply adjustable from 4V to 26V*
- *KERMIT communications program*
- *SXA51 8051/8052 cross assembler*
- *BTK52 BASIC programmers toolkit with tutorial*
- *160 page BASIC Programming Manual by Systronics*
- *RS232 cable*
- *PDK51 tutorial*
- *Optional: aluminum case—$45, monitor/debugger source—$25, battery-backed real-time clock—$39.*
- *All components are manufactured to exacting standards and warranteed for one year.*

PDK51 PLUS includes everything in the PDK51 plus Vers. 3 of our popular BXC518051/8052 BASIC compiler-$800.

Call Now! 508-369-9556 or FAX 508-369-9549

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

#106

PLIX™ HARDWARE X-10" TRANSCEIVER CHIP

Micromint introduces its new Power Line Interface for X-10. PLIX is an 18-pin ASIC chip that automatically handles all the specialized X-10 timing and bit-shuffling between a computer and a TW523 power line module.

- ✖ Complete interface between parallel port and TW523 or PL513 modules
- ✖ Performs all X-10 transmit and receive functions
- ✖ Detects AC power loss
- ✖ Simple interface and timing – operates even in interpreter BASIC
- ✖ Low power – only 1.8 mA @ 5 V

PLIX chip and data sheet	**$20**
100 qty. OEM	**$12**

PLUS SHIPPING

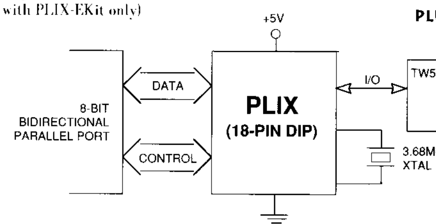
Call **1-800-635-3355**
or Write for a PLIX data sheet

PLIX EVALUATION BOARD

To speed your evaluation, we have produced a low-cost PLIX interface that plugs into a parallel printer port. The PLIX-EKit includes a PLIX chip and data sheet, PCB with all components, application note, and PC-compatible software on diskette.

SPECIAL INTRODUCTORY PRICE
$29.00
PLUS SHIPPING

Add $20 for TW523 X-10 transceiver module (sold with PLIX-EKit only)



MICROMINT, INC.
4 PARK ST., VERNON, CT 06066
(203) 871-6170 FAX: (203) 872-2204

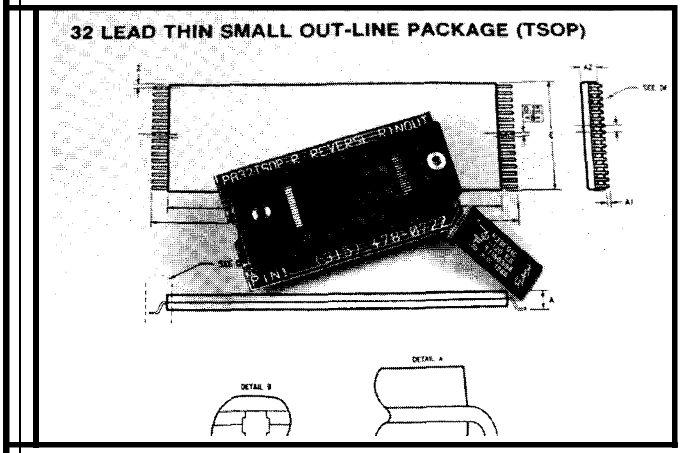
X-10 is a trademark of X-10(USA), Inc.

#119

NEW PRODUCT NEWS

PROGRAM TSOP FLASH MEMORY ON DIP PROGRAMMERS

Flash memories are being designed into an increasing number of embedded systems. The TSOP (Thin Small Outline Package) versions reduce board size and will continue to increase in popularity. In most cases, traditional device programming must still be performed on a programmer with just DIP sockets.


32 LEAD THIN SMALL OUT-LINE PACKAGE (TSOP)
PA32TSOP-R REVERSE PINOUT
(315) 478-0722
DETAIL B
DETAIL A

The **PA32TSOP** from Logical Systems allows existing EPROM and logic programmers with DIP ZIF sockets to support TSOP Flash memories. The adapter accepts the 32-pin TSOP package and plugs into the programmer's 32-pin DIP ZIF socket.

Flash memory sizes of 512 kilobits, 1 megabit, and 2 megabits are supported. A second adapter, the PA32TSOP-R, programs the reverse pinout package of these flash memories. The reverse pinout flash memory simplifies the circuit design of multichip flash memory arrays.

The PA32TSOP and PA32TSOP-R each sells for $95.

Logical Systems Corporation
P.O. Box 6184
Syracuse, NY 13217-6184
(315) **478-0722**
Fax: (315) 4758460

#508

#107

#108

FEATURES

DESIGN CONTEST WINNER

FEATURE ARTICLE

Philip C. Pilgrim

A Single-chip Video Wind Gauge

Just when you thought you knew the limits of Microchip's PIC processor, along comes a project that squeezes even more performance out of it. Find out just why the judges picked this project as last year's contest winner.

When it came time to put together a Circuit Cellar Design Contest entry, I decided to build an instrument that measures wind speed and direction. This device calculates both average and gust velocities and displays its results on a standard TV monitor. The impetus for this project was my wish to answer the following two questions: "Is generating an NTSC video signal using an inexpensive microcontroller and no other ICs possible? If so, will enough processor time remain for the calculation and display of anything useful?" Happily, the answer to both questions is "Yes! ″

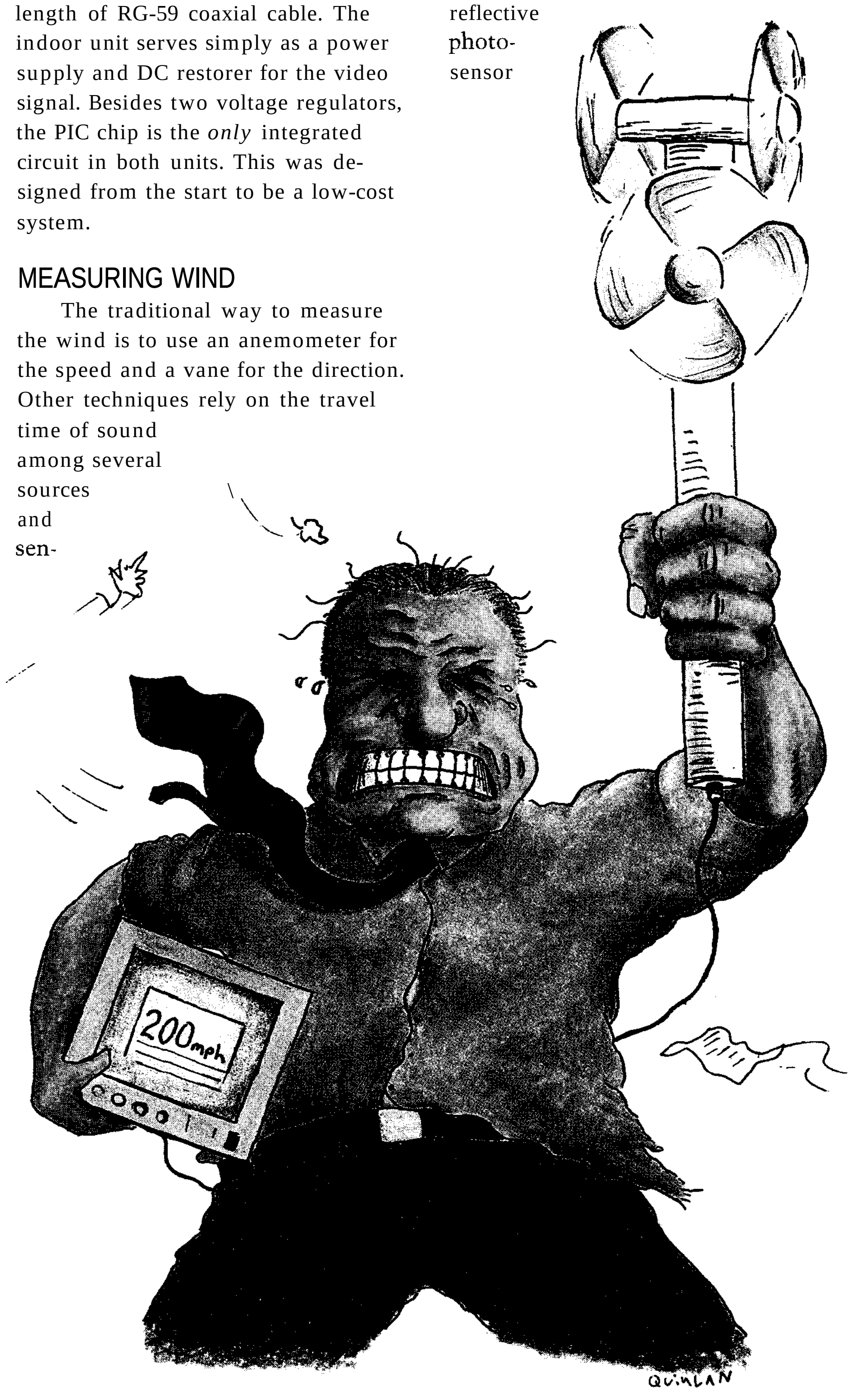
The microcontroller chosen for this experiment is the Microchip Technology PIC16C56. You can obtain this 18-pin wonder in a one-time-programmable form for under $6 each, quantity one. For this amount you get a device that executes 5 million instructions per second from a 1024-word by 12-bit PROM and has 32 8-bit registers, 25 of which are general-purpose. It includes 12 bits of CMOS-level I/O, a real-time clock/counter, and a watchdog timer. What you don't get are interrupts, a big fancy instruction set, a bus interface, or subroutine nesting more than two levels deep, but its speed and elegant command repertoire more than compensate for the chip's otherwise primitive architecture.

The wind gauge consists of two units. The roof unit has four propellers and one PIC chip to measure the wind and generate the video signal. It receives its power from and sends its signal to the indoor unit via a single length of RG-59 coaxial cable. The indoor unit serves simply as a power supply and DC restorer for the video signal. Besides two voltage regulators, the PIC chip is the *only* integrated circuit in both units. This was designed from the start to be a low-cost system.

MEASURING WIND

The traditional way to measure the wind is to use an anemometer for the speed and a vane for the direction. Other techniques rely on the travel time of sound among several sources and sensors or on the cooling rates of heated temperature sensors. I chose to rely on the relative rotation rates of four fan blades oriented in the four cardinal directions.

The fan blade chosen for the job is a 5" plastic affair available from W.W. Grainger as part number 5C169. To test its response to the wind, I mounted it with a bronze bushing so it could rotate freely on a 1/8″ shaft. I installed a reflective photosensor pointing towards the hub face, adhering a semicircular-shaped reflective label upon it. This whole assembly was mounted atop my car so I could rotate it to any horizontal angle with respect to the direction of travel. Then, I programmed a laptop computer to time pulses from the photosensor and determine the rotation rate of the prop.



Having thus outfitted my car as a Research Vehicle, I proceeded to the nearest isolated stretch of highway to run some tests. These tests consisted of nothing more than aiming the prop at a known angle of attack, getting the car up to a known speed, and recording the prop's rotation rate. I completed all tests in both directions on the road to cancel the effects of any real breezes present.

After many hours testing with various speed and angle settings, three results became apparent:

1. At any fixed angle, the rotation rate is directly proportional to wind speed.

2. At any fixed wind speed, the rotation rate is maximum with the prop facing head-on into the wind, dropping to zero at right angles to the wind. The shape of the angular response curve is not a cosine function, as you might expect, but a little more complex, as Figure 1 shows.

3. For a fixed wind speed, the prop is much more efficient (turns faster] with the wind striking it from the front rather than from the rear. This behavior is due both to the extreme curvature of the prop's blades and to a shading effect caused by the structure supporting the shaft. (Initially, I thought that two props mounted at right angles would suffice to build a wind gauge, but the shading effect causes a distorted backside response rendering this design impossible.)

Having determined the speed and direction response of one prop, you can predict the combined response of two props mounted at right angles, as Figure 2 illustrates. Assuming that the wind is between North and East, you can determine both its speed and direction by using this graph. Simply draw a vertical line from the East-facing prop's rotation rate and a horizontal line from the North-facing

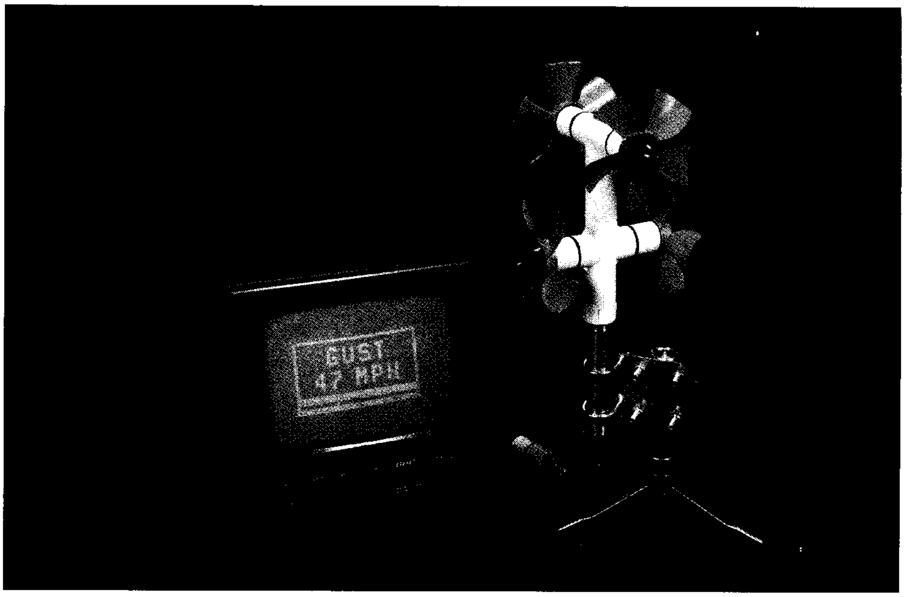


Photo I--The bulk *of the* wind *gauge,* including *the PIC processor, is mounted on the roof. Power to the unit and composite video from the unit are passed through a singe coax cable.*

prop's rotation rate. Where the two lines intersect, you can measure the wind speed and direction from the curvy coordinates inside the quadrant.

For example, if the East prop was rotating at 170 revolutions per second (RPS), and the North prop was spinning at 42 RPS, you could determine that the wind was coming from 70" (or ENE) at 40 miles per hour (MPH). (Had the prop's angular response been a cosine curve, the equal-velocity lines would have been circular and the equal-angle lines would have been uniformly spaced.)

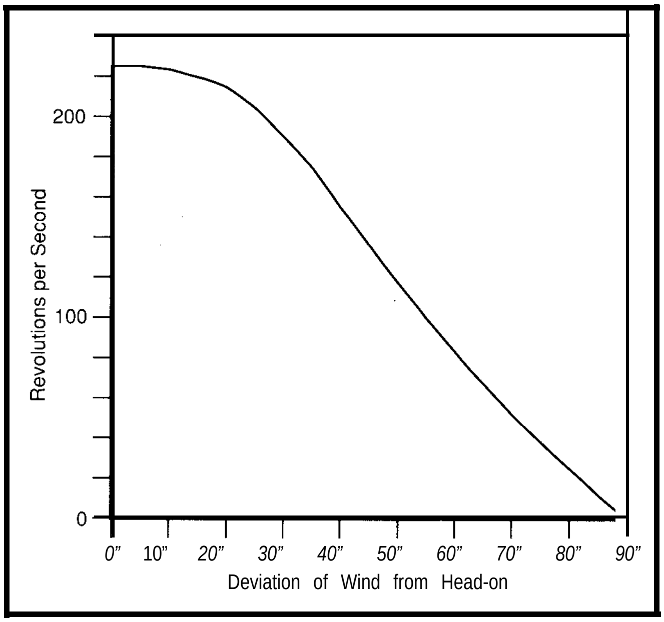


Figure I-As *a propeller is gradual/y turned away from the direction of the wind, the propeller's rotational speed slows. It stops spinning altogether when turned at a right angle to the wind.*

For calculation purposes, I reduced this graph to a table of numbers in the following manner. Assuming the wind is in the quadrant specified, I need only know which of the two props is turning faster to know which half of the quadrant (octant) contains the wind direction (i.e., which of the two cardinal directions is preferred). To refine the direction even more, I take the ratio of the slower rate to the faster rate. For example, a ratio of 20/180 or 0.1111... always corresponds to an angle of 10" from the nearest cardinal direction.

By constructing a table of such ratios, I can write a computer program that looks up the indicated direction, given any ratio measured. In the wind gauge described here, I go one step further. Because the wind gauge only indicates direction in verbal terms (e.g., E, NE, NNE, etc.), the table consists of two ratios. The first corresponds to an angle of 11.25"; the second, to 33.75". These values are the separation angles between N and NNE and between NNE and NE, respectively.

For speed, I need this ratio, plus the rate of the faster prop. Assuming the wind is coming head-on, the speed in MPH is just 0.2222 times the rotation rate in RPS. For other directions, I just calculate the speed from the faster prop as if the wind were head-on and then multiply it by a correction factor. A table of such correction factors can be constructed to be indexed by the low/high ratio. The program would need merely to look up the appropriate factor and apply it to the head-on speed. For example, for a 1: 1 speed ratio (45°), the correction factor would be about 1.67.

Now, how do I determine which quadrant the wind is in! I do so by arraying four props facing outward from a common axis and at right angles to each other. Because the props are more efficient in a frontal wind than from behind, I can always tell from two opposed props which semicircle the wind is in by which prop is turning faster. Two such sets of opposed props at right angles yield the correct quadrant as the intersection of two semicircles. Also, this step is done without even measuring a prop's direction of rotation-just its speed.

MECHANICAL HARDWARE

I constructed the wind gauge from off-the-shelf components. The only custom fabrication requiring more than a drill press or saw is slotting the stainless steel shafts for snap rings. Most machine shops will be happy to do this step for a nominal fee. The primary structural components are 3/4″ PVC pipe and fittings. Photo 1 shows the completed wind gauge, from which you can infer the gross structure.

The top two props are the North/South set; the lower two, East/West. I've housed the circuitry for this unit in the brass lower pipe with cables running up to the prop rotation sensors. Everything is sealed against moisture with PVC cement, 0 rings, Teflon tape, or epoxy. You can mount the unit using a clamp in a variety of ways. I made the

Can you justify owning a personal programmer?

You can have one tomorrow on approval - and then decide.

$795

Size: 7.3 x 4.4 x 1.8 inches
Weight: 18 ounces

DATAMAN S4

Dataman S4 is Europe's most widely-used PROM programmer and emulator. It has become the preferred tool with engineers who choose the best, because it offers a unique combination of speed, convenience and capability.

Convenience S4's size and weight allows it to be carried easily wherever you need to work; you are not tied to the bench. It is battery powered, and can program up to a thousand PROMs on a single charge. Normal operation can continue while recharging, which takes less than an hour. A lithium cell protects your data and library, should the main battery ever run flat.

Capability S4 can program almost anything that will fit into its socket, including UV, OTP, EE and Flash devices up to 8 megabits. It comes with 1 Mbit (128K bytes) of internal memory as standard, but you can easily upgrade this to 4 Mbit (512K bytes). Its RS232 port supports file transfer in Binary, ASCII, Intel, Motorola and Tek formats. Because it loads its library and setup data from a device in its socket, you can program your preferred setup into a ROM, so that if anyone else uses your S4 you can quickly reset it when you get it back - IF you get it back.

ROM Emulation The modify-program-run-crash cycle can be really tedious, especially when you run out of fresh PROMs and have to wait for erasure. But by plugging the S4's emulation lead into your target system in place of the ROM, you can test your code before committing it to silicon. Connect the emulation cables to your target system, download your assembled code to S4, press the EMULate key, and your product runs the program - it's as easy as that. The only PROM you ever need to program is the last - correct - one.

***Software** Development* S4's memory emulation is an inexpensive alternative to a full ICE, and it works with any microprocessor. Many engineers prefer it, because their prototype runs under conditions more like the real world. S4 can emulate ROMs up to 512K bytes, but unlike a ROM, you can also write to it. If you put your variables and stacks in the memory area you're emulating, you can inspect and edit them as easily as your code.

After-sales service S4's device library is constantly being upgraded, and you can download the latest version at any time, free of charge, from our 24-hour bulletin board. Technical support is freely available by phone and modem. S4 comes with a 3-year full parts and labor warranty.

Complete and ready to go S4 comes complete with everything you need to go to work right away, including the charger, emulation lead, write lead, library ROM and software utilities. These include a dedicated MS-DOS driver with built-in help for all of S4's functions, which can communicate with it at speeds up to 115,200 baud.

Money-back-guarantee If you call us today, we will put a Dataman S4 in your hand tomorrow, if a method is available. Use it for a month. If you cannot justify owning an S4, just return it and get your money back, no questions asked.

DATAMAN
BOO-DATA-336
(800) 328-2336

VISA MasterCard

Dataman Programmers, Inc.

US: 22 Lake Beauty Drive, Suite **101,** Orlando, FL 32806 (407) 649-3335, FAX (407) 649-3310, BBS (407) 649-3159 24 hr
UK: Station Road, Maiden Newton, Dorset DT2 OAE, England 0300-20719, FAX 0300-21012, BBS 0300.20195 24hr

#109

clamp in the picture from U bolts and a piece of 1/S" aluminum.

Figure 3 shows the prop assembly in a little more detail. A reflective optosensor (Omron EE-SY101}, which fits snugly in a #21 hole drilled in the end of a flat-profile (not domed) pipe cap, senses prop rotation. It has an optimum sensing distance of 1 mm, so you should mount it flush with the end of the cap. Because the sensor has a flat on one side that leaves a gap running the length of the hole, you need to seal it from behind with epoxy.

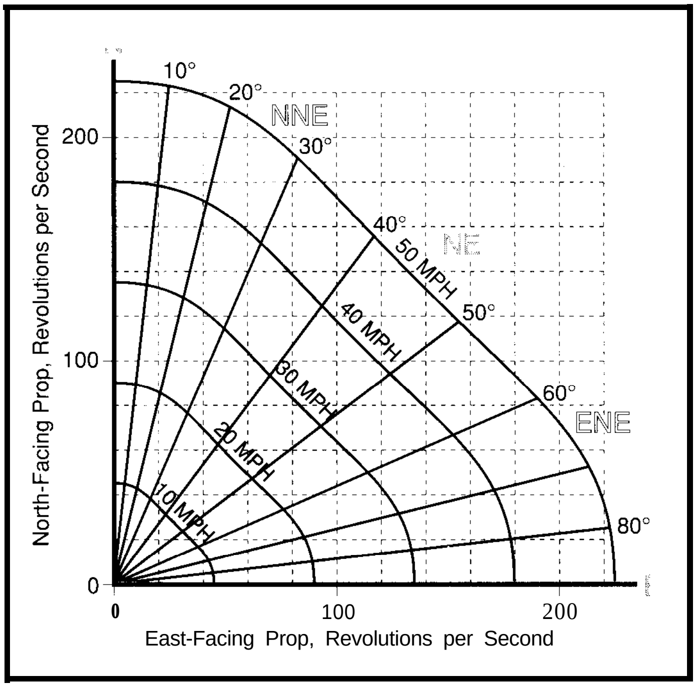
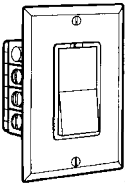
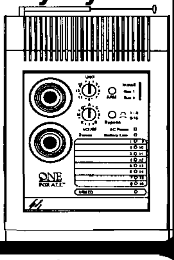


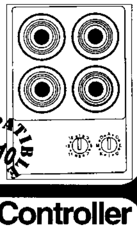
Figure P-Given *the speed of a pair of propellers mounted at right angles to each other, it's a simple matter to figure out the incident wind speed and direction.*

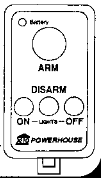
This sensor looks at the inside face of the prop hub, where I've adhered a reflective semicircular-shaped label over a flat, black background. As the prop rotates, the optosensor sees an alternating black and reflective pattern and outputs a corresponding square-wave signal. Paint the end of the pipe cap flat black to keep sunlight from bouncing in from the edge and saturating the photosensor. Also, the specified fan blade is color-coded gray to indicate counter-clockwise rotation. Do not use white (clock wise) fan blades because too much light will get through the plastic.

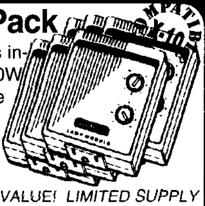
The combination of bronze bushing (from any bearing supply] and stainless steel shaft (from Winfred Berg Inc.) forms a free-spinning but close-tolerance fit. This fit is essential to prevent chattering and resonances, which would destroy the gauge's accuracy. The shaft is press-fit into a hole drilled in the center of the pipe cap. Install it with the fan blade, the washers, and the snap ring in place, along with a thin piece of plastic, which you remove later. This arrangement establishes some necessary end play and gives the nylon washers room











to swell when they get wet. The shaft may be epoxied from behind for permanence.

You could cement this whole assembly to the tee (or the cross for the East/West props), but doing so would preclude any later disassembly. By using a spring and a couple 0 rings as shown in Figure 3, you preserve the watertight integrity and you can still get the device apart. Cones may be added to the props in various ways to dress them up, but are not essential to the operation. (Indeed, you risk unbalancing the prop or affecting its calibration by adding them.)

The cones in Photo 1 were hand-laid from fiberglass in a plaster mold made from a Silly Putty egg, sanded, and painted black. I simply stuck each one in place for the photo using butyl tape. (So much for photo secrets.) A better technique would be to extend the shaft well beyond the snap ring groove and attach a cone to the shaft, leaving a gap between it and the outside face of the prop hub. The cone would not turn, so it would not affect the prop's balance, although its presence in the wind stream might still alter the prop's angular response curve.

I housed the indoor unit in two 1/2″ PVC pipe caps joined with a stub of PVC pipe (see Photo 1). One of the exit cables goes to the monitor, the other goes to a wall transformer-type power supply.

VIDEO INTERFACE

To display information on a standard video terminal, the microcontroller must generate a signal that conforms (more or less) to the NTSC standard. This standard defines things such as relative voltage levels for various parts of the signal and their timing. The wind gauge generates a noninterlaced, black-and-white signal with a negative-going composite sync. Figure 4 illustrates such a signal.

In those systems with just black and white and no grays in between, the signal has three levels. The sync level is 0 volts, the blanking level is 0.28 volts, and the white level is 1 volt. These levels form a l-volt peak-to-peak (p-p) signal which must be able to drive a 75-ohm ground-referenced load. (In many cases, a fourth or black level is defined slightly above the blanking level, and it defines true black. The blanking level is then referred to as *blacker than black. You* don't need this voltage level here.)

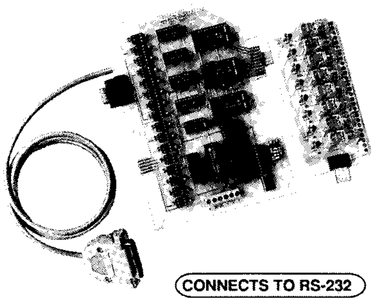
The signal's second major feature is a horizontal sync pulse that occurs every 63.5 µs. This pulse establishes the beginning of each line and allows the monitor's horizontal oscillator to lock on to the signal. The actual period of the horizontal sync need not be exactly 63.5 µs because all monitors have some built-in latitude. However, whatever time you chose, it must not vary the slightest from sync to sync, or an unsteady picture will result.

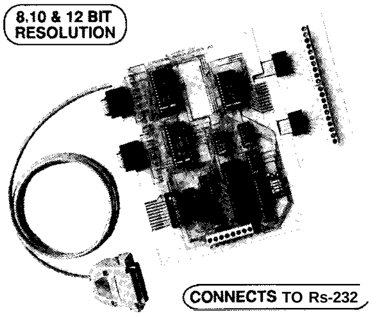
The third major feature of this signal is the vertical sync. This sync consists of three horizontal intervals in which the signal stays at sync level for most of the interval. (The brief positive-going pulses are called *serrations* and maintain horizontal sync during the vertical sync.)

In interlaced systems, the vertical sync occurs after each occurrence of 262.5 horizontal lines, or at a rate of 60 Hz, and locks the monitor's vertical oscillator to the signal. Because noninterlaced signals don't use half lines, they can use 263 lines as the interval (59.89 Hz). Also, in interlaced systems, an interval of *equalizing pulses,* or horizontal syncs at half the usual spacing, precedes and follows the vertical sync. Those aren't necessary here. The vertical sync is followed by the vertical blanking interval, which consists of at least 15 horizontal intervals with sync pulses but no signal above the blanking level.

Finally, in those lines having visible pixels, the signal will make excursions to the white level to display them on the screen.

The signal as defined will drive most composite sync monitors. You may also apply it to newer TV sets with a video input. It cannot be connected to a TV's antenna terminals, though, except through an RF modulator. Picture quality on a good monitor will be excellent; however, some TVs do poorly with high-contrast signals in which the white level is a

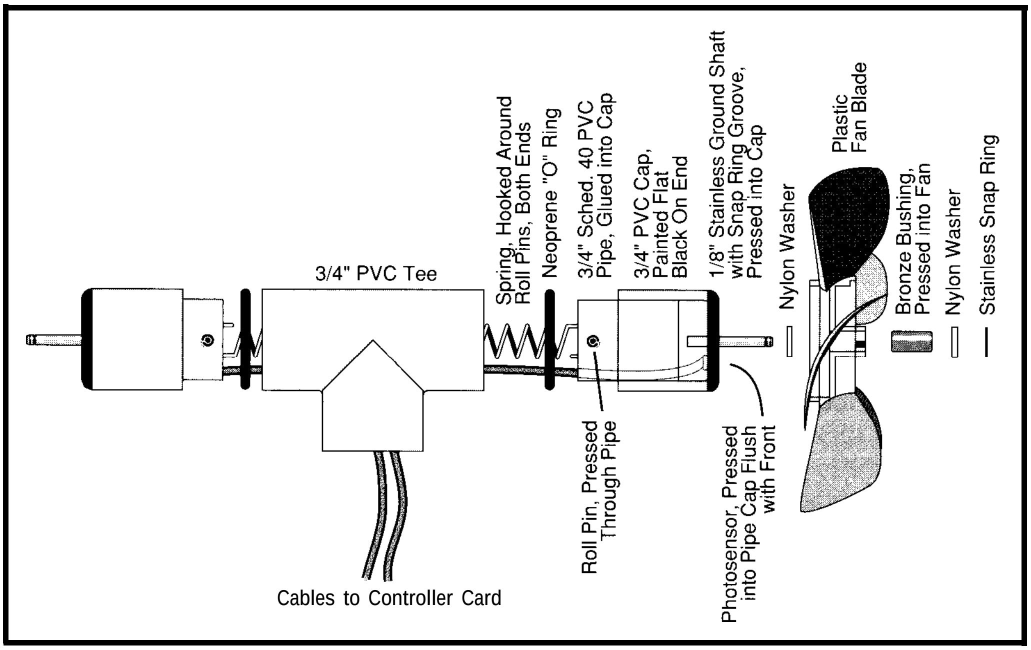




full volt. In these cases, reducing the white level may be necessary.

ELECTRONIC HARDWARE

Figure 5 shows a schematic of both the indoor and roof units of the wind gauge. A small, wall-plug-style AC-to-DC converter can power the indoor unit. The converter should have good filtering or you'll need to use a much larger value for C 1. The indoor unit provides operating current to the roof unit through 75-ohm resistor Rl and the RG-59 coax cable. The roof unit, in turn, generates a 2-volt p-p video signal that modulates the supplied power. Rl serves as both the impedance termination for the indoor unit and as a pull-up resistor for the line driver in the roof unit. The returning video signal is AC-coupled to the base of Ql. Diodes Dl-D3 serve to clamp the negative-going sync tips to 0.6 volts. Transistors Ql and Q2 act together as a high-beta NPN follower known as a Sziklai connection. This configuration is similar in purpose to a Darlington, except that the base-emitter threshold is only one diode drop instead of two. This follower then drives the video monitor through resistor R5, yielding a 1 -volt p-p signal into a 75-ohm load.



3/4" PVC Tee
Spring, Hooked Around Roll Pins, Both Ends
Neoprene "O" Ring
3/4" Sched. 40 PVC Pipe, Glued into Cap
3/4" PVC Cap, Painted Flat Black On End
1/8" Stainless Ground Shaft with Snap Ring Groove, Pressed into Cap
Nylon Washer
Plastic Fan Blade
Bronze Bushing, Pressed into Fan
Nylon Washer
Stainless Snap Ring
Roll Pin, Pressed Through Pipe
Photosensor, Pressed into Pipe Cap Flush with Front
Cables to Controller Card

Figure 3—*Virtually everything in the wind* gauge's propeller *assembly is available off-the-shelt.*

You can use the cheapest 75-ohm coax available to connect the indoor unit to the roof unit. Cable quality isn't critical because no RF is involved. However, lightning protection is critical. You must ground any metal supports and include a well-grounded lightning arrestor in the installed cable before it enters any building. Check your local building and electrical codes for other requirements. While a direct lightning strike will certainly take out the roof-mounted electronics, there's no sense risking your video equipment, not to mention your house!

From the incoming cable, power to the roof unit passes through a low-pass filter consisting of R6 and C5. The voltage at this point will be about 7.5 volts. It's regulated down to 5 volts by VR2, a 78L05, which powers the PIC microcontroller. The PIC includes an external reset that in some systems can simply be tied to Vcc. In cases where the power-on voltage ramps up slowly (e.g., a low-current wall transformer with a big filter capacitor), you must use the external reset circuit (R13, C7, D4). Also, the RTCC input should be tied high.

The PIC uses a 19.6608MHz crystal. There's no particular reason for this exact frequency-it's just the closest thing I had in stock to 20 MHz, the fastest the PIC will go. However, if you should deviate from 19.6608-MHz, you will have to change some constants in the program regarding video timing.

The PIC's two ports are well divided along functional lines. Port A is the video output, and port B inter-

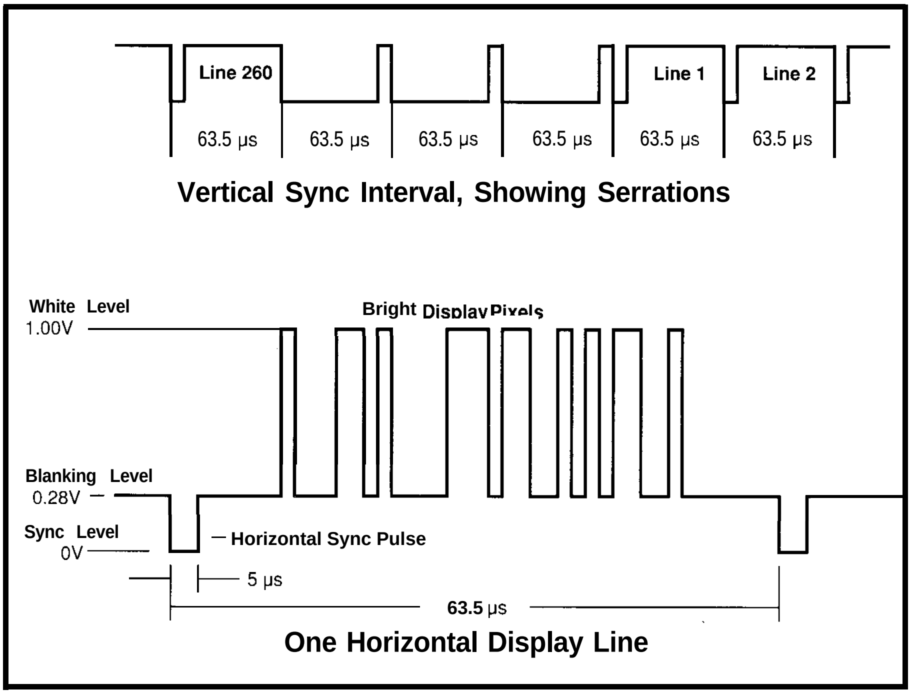
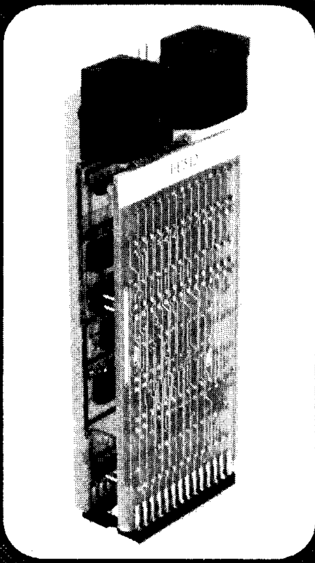


Figure 4—*The* wind gauge *generates* a more-or-less *NTSC-compatible, noninterlaced,* black-and-white signal *with* a negative-going *composite sync.*



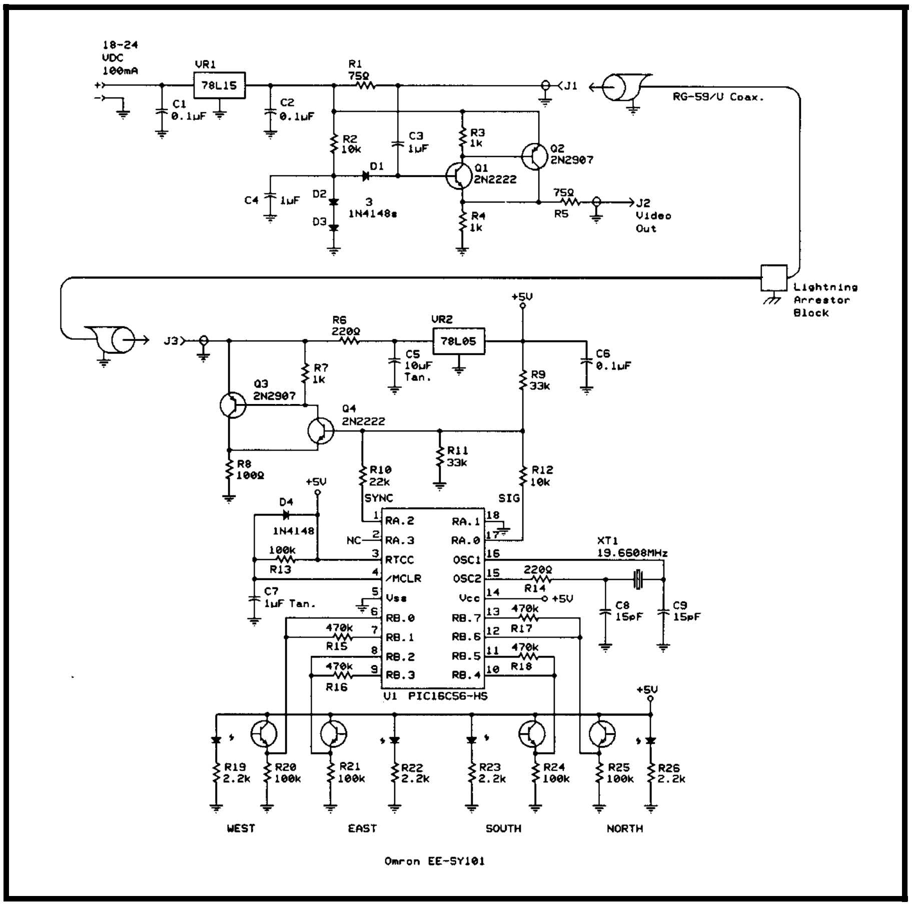


Figure **5**—*The indoor unit (fop) provides operating current to the roof unit (bottom) through an FIG-59 coax cable and a 75-ohm resistor, R1 (which also serves as the impedance termination for the* indoor *unit and as a pull-up* resistor *for fhe line* driver in *the* roof *unit).*

faces to the four optosensors. Looking first at port B, you see that even numbered port lines connect directly to the phototransistors, while odd-numbered lines connect through 470k resistors. The even lines are inputs and the odd lines are outputs. The outputs track the inputs under program control and provide a slight positive feedback or hysteresis to the sensors. This feedback in turn provides a good measure of noise immunity to the inputs, eliminating the need for complicated debouncing schemes.

I used the two lines of port A as video outputs. Bit 0 is the signal (0 volts is white; 5 volts is black) and bit 2 is the sync (5 volts is sync; 0 volts is no sync). These two signals are summed through resistors R10 and R12 at the base of Q4, which is nominally biased via R9 and R11 to 2.5 volts. Q3 and Q4 form another Sziklai connection, but this time as an inverting amplifier. The output impedance of this amplifier, when considering R1 as the load, is equivalent to the parallel combination of R8 (100 ohms) and R6 (220 ohms], or 68.75 ohms-close enough to 75 ohms. Bit 1 of port A is an input and must be grounded (more on this requirement in the next section), and bit 3 is an output with no connection.

Photo 2 shows the prototype circuit board for the roof unit. I housed it in the brass support pipe of the wind gauge. Line this pipe with a rolled-up piece of plastic or paper to insulate the wall of the tube from the circuitry. The brass will serve as a shield against RFI radiated by the board. Stuffing a piece of copper scouring pad (e.g., Chore Boy) into the top of the pipe where the sensor cables exit will also help. However, don't use bronze or steel wool because the tiny pieces of metal shed by these products could fall onto the circuit board. You can attack conducted RFI using ferrite beads, both on the power/video line and on the sensor cables. Copper braid over the sensor cables might also work.

That’s it hardware-wise. All the real work is done in firmware, as the next section reveals.

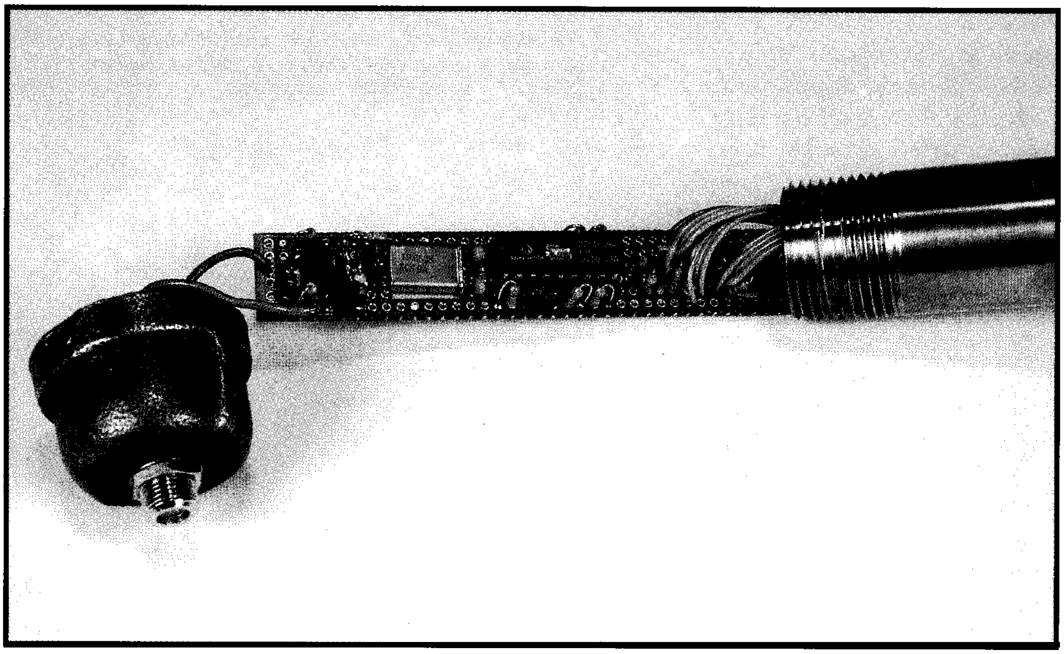


Photo 2—*Except* for the power *supply and level shifters, all the wind gauge electronics fit inside the shaft of the roof unit.*

FIRMWARE

Listing 1 shows selected examples from the source program. (A complete assembly listing is available on the Circuit Cellar BBS.) I wrote the program for the Parallax PIC cross-assembler, and it uses Parallax’s mnemonics almost exclusively. This program uses 963 of the PIC16C56’s 1,024 available instruction words and all of its registers.

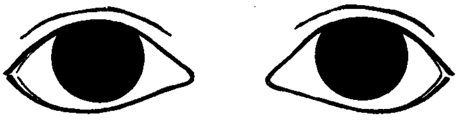
To understand why the program is organized as it is, I need to explan of some of the PIC’s peculiarities. First, the PIC’s program space is divided into two 512-word pages or segments. Intersegment jumps or calls must be prefaced by setting a segment bit in the status register, so grouping callers and callees in the same segment has some advantages. Second, only the first 256 addresses in each segment can be the destinations of subroutine calls, so I located all subroutines in these two blocks. Third, in the PIC, tables are implemented as subroutines, with each entry being a return instruction that loads a constant into the accumulator (W register), so I also had to place them in the first half of each page. Finally, because the PIC’s stack is only two levels deep, I had to place much of what normally would be subroutine code as in-line code.

With that out of the way, I can describe in a functional manner what the program actually does. At the grossest level of abstraction, the program performs three tasks:

1. It periodically counts the rotations of the four propellers for a fixed amount of time.

2. From these counts, it calculates an instantaneous speed and direction, keeps track of gusts, and maintains a running average speed and direction.

3. It displays an outlined box on the video screen (see Photo 1). Inside this box, it displays the instantaneous speed as a continuously animated pointer on a horizontal scale near the bottom. Alternately, at 4-second intervals, it displays the recent highest (gust] speed as an integer in MPH, then the direction (up to three letters; e.g., ESE) and speed (as an integer in MPH) of the running average wind. These it displays digitally in large characters.



(When displaying the average wind, the direction characters replace the word GUST on the display shown in Photo 1. If there is no wind, the direction is left blank.) In all cases, the maximum wind speed displayed is 99 MPH. (Whether this gauge would even survive hurricane-force winds or not is another question.)

Everything in the program revolves around keeping the video signal alive because the timing of the video output is critical. In traditional computer systems having video output, a separate chip provides all the video timing and refreshes the display from a special section of RAM. This arrangement leaves the processor free to do its work with little urgency regarding the video. Indeed, all it has to do to display anything is stuff characters or pixels in video RAM, and the video controller does the rest. However, I have not afforded myself this luxury.

In fact, not only must the program set and reset every bit of signal and sync at the proper moment, but there is no video RAM (just those 25 registers, remember) to store those bits. Each displayed pixel must be computed virtually on the fly! And this is in addition to doing the calculations necessary to display something useful in the first place.

The heartbeat of this program is the horizontal sync. At 19.6608 MHz (4.9152-MHz instruction rate), this signal must occur every 312th instruction come rain, shine, or division by zero. The real-time clock/counter (RTCC) handles the timing for the horizontal sync. This timer is programmed to count up at the instruction rate. After reaching 255, it rolls over to zero to start another cycle. The RTCC can be read and written (writing causes a two-cycle count delay) but cannot cause any interrupts; it must be polled.

To get 3 12 counts out of a modulo-256 counter, the RTCC must pass through zero twice. On the second passage, the program is assumed waiting to reload the counter with a new initial count and to output a pulse on SYNC. One subroutine, HSY NC (see Listing la) performs the

Listing I-A/though *selected examples are shown here, some of the tasks the source program performs include the calculation of instantaneous speed and direction.*

```
; Section la: Horizontal Sync Routine

HSYNC    JNB   SAMPLE,PROPOK    ;Sample propellers?
         MOV   W,<<RPROP        ;Yes: Get prop input bits
         NOT   W                ;Invert
         AND   W,RPROP          ;AND with previous bits
         MOV   DUR,W            ;Save in DUR
         SNB   DUR.7            ;Inc props on falling edges
         INC   NPROP            ; and saturate revs at 7FH
         SNB   NPROP.7
         DEC   NPROP
         SNB   DUR.5
         INC   SPROP
         SNB   SPROP.7
         DEC   SPROP
         SNB   DUR.3
         INC   WPROP
         SNB   WPROP.7
         DEC   WPROP
         SNB   DUR.l
         INC   EPROP
         SNB   EPROP.7
         DEC   EPROP
PROPOK   RL    RPROP            ;Record bits in hysteresis outputs

HSYNCO   CALL  LOCSYNC          ;HORIZONTAL SYNC ONLY ENTRY POINT
         MOV   RTCC,#HTIME      ;Reset RTCC for new line
         SETB  SYNC             ;Set sync output
         MOV   DUR,#7
         DJNZ  DUR,$
         CLRB  SYNC             ;Reset sync
         RET

LOCSYNC  MOV   W,#-LOCDLY>      ;WAIT FOR RTCC TO CROSS ZERO
                                (This waits for horizontal sync time
                                or for first 256-count of horizontal
                                interval to pass.)

LOCATE   MOV   W,RTCC-W         ;RETURN AT EXACT TIME RE: RTCC, PLUS LOCDLY.
         MOV   DUR,W            ;DUR holds 256-delay time, but loop is only
         AND   W,#3             ;accurate to four instructions. Use the JMP
         JMP   PC+W             ;PC+W for DUR MOD 4
         NOP
         NOP
         NOP
         MOV   W,#4
LOCLP    ADD   DUR,W            ;Loop is four instructions long, so add
         JNC   LOCLP            ; four until DUR overflows
         RET
```

```
; Section lb: Vertical Sync and Main Video Loop.

VSYNC    CLR   WDT              ;Reset watchdog timer
         MOV   W,#001011B       ;Set up internal 1:l RTCC, 1:8 WDT
         OPTION
         MOV   W,#0010B         ;Set up RVID.l input; others output
         TRIS  RVID
         MOV   W,#01010101B     ;Set up RPROP.0, .2, .4, .6 input;
                                     others out.
         TRIS  RPROP
         INC   FIELD            ;VERTICAL SYNC, inc FIELD count [ 4 LINES]
         MOV   REP,#3           ;Three lines of vertical sync
```

(continued)

Listing 1-continued

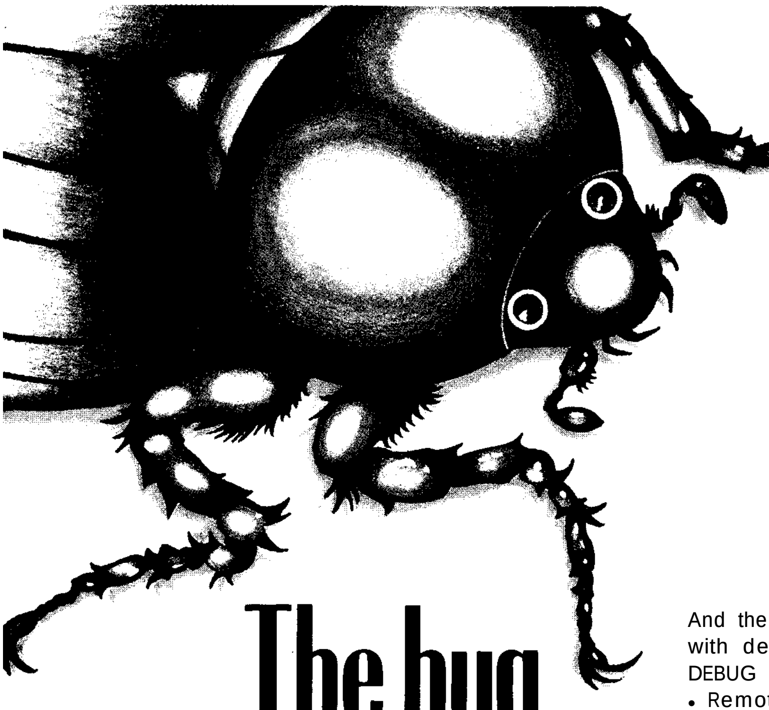
```
DOVS    CALL  LOCSYNC          ;Wait for horizontal sync time
        MOV   RTCC,#HTIME
        SETB  SYNC             ;Set sync bit
        CALL  LOCSYNC          ;Let first 256 pass
        MOV W,#-25-LOCDLY> ;Wait until just before next horiz.
        ;                           sync time
        CALL LOCATE
        CLRB  SYNC             ;Reset sync bit
        DJNZ  REP,DOVS         ;Back for more
        CALL  HSYNCO           ;Do a real horizontal sync
        CALL  LOCSYNC          ;Let first 256 pass

;Blank. Execute foreground task in pieces (see text)       [65 LINES]
;Draw horizontal bar (top of box).                         [ 6 LINES]

;Draw left side of box, then blank, then right side.       [12 LINES]
;Compute the direction characters. This occurs whether
;direction is to be displayed or not and is repeated
;twelve times. (It only needs to be done once, but less
;code is used doing it twelve times.)

;Show the direction (e.g., "ESE") or the word "GUST"       [36 LINES]

;Blank, except for left and right borders. Format          [12 LINES]
;the speed from the BCD number in register MPH.

;Show the speed and the letters "MPH"                      [36 LINES]
```

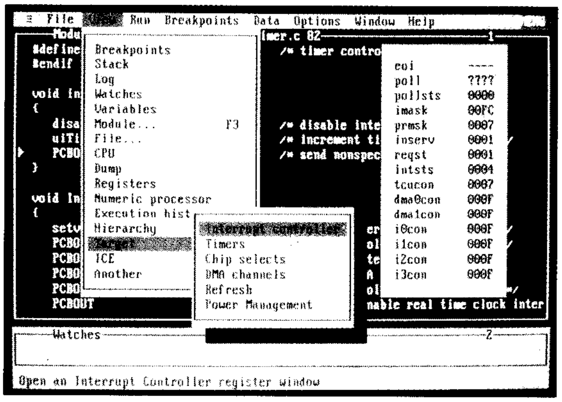
(continued)

polling, waiting, and SYNC-pulsing chores. It also samples the props, but more on this feature later. At the beginning of each horizontal line, the program calls HSYNC. HSYNC calls LOCATE via LOCSYNC, which waits for the RTCC to reach zero. On the return from LOCATE, the RTCC is reloaded, the SYNC bit is set, then reset after 5 us, and HSYNC returns.

The whole main program is just one grand loop that executes once every 1/60 of a second. After power-up initialization, this loop begins by clearing the watchdog timer and reinitializing some configuration registers. (Microchip recommends this step in case noise obliterates their contents, but the program otherwise continues to run.) The program then generates a vertical sync sequence. From there it does 65 blank lines, at which point the top of the display box is reached.

Here it draws a horizontal bar, then proceeds to display everything that goes into the box, one line at a





time. Each line begins with a call to HSYNC, followed by some twiddling of the SIG line to lighten some pixels here and there. After that, it's ready to do the bottom bar of the box and then 46 blank lines. A jump instruction then transfers control back to the beginning for another vertical sync and another field of video, ad infinitum. I've summarized the organization of this loop in Listing lb.

A few comments are in order about how the program displays various features. The SHOLINE routine displays the large characters (see Listing lc). This routine takes a byte from an indirectly referenced register and rotates it left one bit at a time into the carry bit. The carry bit is then rotated into left port A, whose bit 0 is the SIG output. This explains why bit 1 must be a grounded input. Because the contents read from a port are always the levels on the port pins and not the contents of the output register, this fixed zero level will block any rotates of nonzero bits into bit 2, the SYNC bit. Pixels generated in this fashion have a minimum width of two instructions, or 0.4 us.

GETLINE (see Listing Id) provides the bit patterns displayed by SHOLINE. This subroutine takes a base address (the character identifier) plus the value of REP (the row number) and returns the bit pattern for the selected row of the chosen character. With WIND GAGE's 8 x 12-pixel characters, 21 characters can be defined this way—plenty for this application.

Calls to LOCATE, the same routine used by HSYNC, establish the starting positions of these characters in each horizontal line (as well as the horizontal positions of most other features). LOCATE takes one argument, the horizontal position, and polls the RTCC to determine how long to wait for this position to occur. It then waits that length of time and returns to the caller.

Fixed features, such as the ruler bars and the "0 50 100" legend, are done in-line by code written just for that purpose. Listing le shows how the legend is displayed. With this technique, pixels one instruction time wide are possible, which allows finer

Listing 1-continued

```
;Blank, except for left and right borders                 [12 LINES]

;Horizontal bar (bright area above scale)                  [6 LINES]

;Show 50s ruler bars                                       [ 3 LINES]
;Show 10s ruler bars                                       [ 3 LINES]
;Show 5s ruler bars                                        [ 3 LINES]

;Blank, except for left and right borders.                 [1 LINE ]
;Calculate position of dial pointer as well as gust speed

;Draw the dial pointer in its proper position              [ 6 LINES]
;Blank, except for left and right borders                  [1 LINE ]
;Horizontal bar (bright area between dial pointer and      [ 3 LINES]
;"0 50 100" legend)

;Draw the "0 50 100" legend below dial                     [ 5 LINES]
;Horizontal bar (bottom of box)                            [ 3 LINES]
;Blank (bottom of screen)                                  [46 LINES]

        JMP   VSYNC            ;Go back for another display field
```

```
; Section lc: Character Row Display Routine.

SHOLINE RL    INDIRECT         ;SHOW ONE ROW OF EIGHT PIXELS ON VIDEO OUT
        RL    RVID
        RL    INDIRECT         ;Rotate pixels into carry,
        RL    RVID             ; and from carry into SIG (RVID.0)
        RL    INDIRECT         ;Gnd on RVID.1 prevents rotation into SYNC
        RL    RVID             ;Each pixel is two instr (407 ns) wide
        RL    INDIRECT         ;Code is in-line to avoid loop overhead
        RL    RVID
        RL    INDIRECT
        RL    RVID
        RL    INDIRECT
        RL    RVID
        RL    INDIRECT
        RL    RVID
        RL    INDIRECT
        RL    RVID
        INC   FSR              ;Point to next 8-pixel group
        RET
```

```
; Section Id: Character Row Select Routine.

GETLINE
      ADD  W,REP               ;GET ROW OF EIGHT PIXELS. Character is in W
      JMP  PC+W                ; row is in REP. Bottom row is row one
      RETW 0                   ;There is no zero row
      RETW 83H,1,39H,39H,39H,39H,39H,39H,39H,39H,1,83H           ;[0]
      RETW 81H,81H,0E7H,0E7H,0E7H,0E7H,0E7H,0E7H,0E7H,87H,0C7H,0E7H;[1]
      RETW 1,1,3FH,1FH,8FH,0C7H,0E3H,0F1H,0F9H,39H,1,83H         ;[2]
      RETW 83H,1,39H,0F9H,0F9H,0E3H,0E3H,0F9H,0F9H,39H,1,83H     ;[3]
      RETW 0F3H,0F3H,0F3H,1,1,33H,33H,13H,83H,0C3H,0E3H,0F3H     ;[4]
      RETW 83H,1,39H,0F9H,0F9H,0F9H,1,3,3FH,3FH,1,1              ;[5]
      RETW 83H,1,39H,39H,39H,1,3,3FH,3FH,39H,1,83H               ;[6]
      RETW 9FH,9FH,9FH,9FH,8FH,0C7H,0E3H,0F1H,0F9H,0F9H,1,1      ;[7]
      RETW 83H,1,39H,39H,39H,83H,83H,39H,39H,39H,1,83H           ;[8]
      RETW 83H,1,39H,0F9H,0F9H,81H,1,39H,39H,39H,1,83H           ;[9]
      RETW 39H,39H,31H,31H,21H,21H,9,9,19H,19H,39H,39H           ;[N]
      RETW 83H,1,39H,0F9H,0F9H,81H,3,3FH,3FH,39H,1,83H           ;[S]
```

(continued)

detail at the expense of large amounts of code.

Depending on the required position at the moment, a variable argument in the call to LOCATE easily accommodates animated features like the moving dial pointer. This position has a resolution of one instruction.

The main program loop, which displays one field of video each time through, expects the parameters it displays to have already been calculated. But where do these calculations occur? The answer lies in video lines 1 through 65. These lines are blank and there is a lot of idle time available to do some real work. However, considering all the HSYNC appointments you have to keep, the time is only available in tiny chunks. The solution is a form of cooperative multitasking, which the PIC makes particularly easy.

All subroutine returns in the PIC simultaneously load a constant of the programmer's choice into the W register. If a task could be broken into a series of short subroutines, these returned values could indicate to the calling program which subroutine to call next, forming a chain of execution as long as needed to get any job done.

In the example shown in Listing 1f, suppose the register NEXT always contained a number corresponding to the section of routine MAIN to execute next. You could then have MAIN invoked repeatedly inside the loop as shown. As you can see, the routine MAIN proceeds quite independently of the code in LOOP, even having conditional branching. Of course, you would have to be careful of those registers used by both foreground and background tasks to maintain their integrity.

WINDGAGE uses this technique to access the foreground routine. The only complicating factor is that the NEXT value is four bits long and must share a register with another 4-bit value. However, the principle is exactly the same as in the example. The foreground routine does the real work of converting prop rotations to wind speed and direction, averages, gusts, and the like. It also provides the parameters that the grand video loop expects to have ready on time.

Listing l-continued

```
     RETW 1,1,3FH,3FH,3FH,7,7,3FH,3FH,3FH,1,1                    ;[E]
     RETW 93H,93H,1,1,29H,29H,29H,39H,39H,39H,39H,39H            ;[W]
     RETW 39H,39H,39H,39H,29H,29H,1,1,11H,11H,39H,39H            ;[M]
     RETW 3FH,3FH,3FH,3FH,3,1,39H,39H,39H,39H,1,3                ;[P]
     RETW 39H,39H,39H,39H,39H,1,1,39H,39H,39H,39H,39H            ;[H]
     RETW 83H,1,39H,39H,21H,21H,3FH,3FH,3FH,39H,1,83H            ;[G]
     RETW 83H,1,39H,39H,39H,39H,39H,39H,39H,39H,39H,39H          ;[U]
     RETW OCFH,OCFH,OCFH,OCFH,OCFH,OCFH,OCFH,OCFH,OCFH,OCFH,3,3  ;[T]
```

```
; Section le: Routine to Draw the "0 50 100" Dial Legend.
;
DRWLGND ADD  W,REP          ;DRAW '0 50 100' PIXELS. REP is row number,
        JMP  PC+W
        JMP  TBO            ;Top and bottom of '0'
        JMP  MIDO           ;Middle of '0'
        JMP  MIDO                 "
        JMP  MIDO                 "
        JMP  TBO            ;Top and bottom of '0'

        JMP  TB50           ;Top and bottom of '50'
        JMP  BM50           ;Bottom middle of '50'
        JMP  MID50          ;Middle of '50'
        JMP  TM50           ;Top middle of '50'
        JMP  TB50           ;Top and bottom of '50'
        JMP  TB100          ;Top and bottom of '100'
        JMP  MID100         ;Middle of '100'
```

[continued)

The routine begins by waiting for the FIELD count (a register incremented at each vertical sync) to become zero, modulo-64, which it does about once a second. It then clears a set of four registers used as prop rotation counters, and sets a bit telling HSYNC (see Listing la) to sample the props. During the next 34 fields, HSYNC looks for falling edges on the prop sensor inputs, incrementing the appropriate counter each time one occurs. It also performs the hysteresis function simply by rotating port B left (i.e., rotating the input bits into the hysteresis bits).

After the sample time is over, the sample command bit is cleared. Then the wind quadrant is determined by which props are spinning fastest. Registers NPROP and EPROP (the North and East rotation counters) are then assigned the North and East components (possibly negative) of the rotation speeds.

Every fourth sample is used to update the moving average. The North and East prop components of the average are stored in registers AVGN and AVGE, with 4-bit fractional parts in register AVGFRAC. Each component of the average is calculated using the equation

AVG := AVG+(PROP-AVG)/16

When the time comes to display the average wind, AVGN and AVGE are used to calculate speed and direction insteadof NPROP and EPROP. Now, averaging wind components that are not cosine responses to direction is not exactly kosher, because these are not really the x and y components of velocity. However, this average involves a short period of time, remaining valid as long as the wind direction doesn't change radically over its half-life.

You may wonder why I bother with the average at all. I asked a weather consultant this question, as part of finding out just what is meant when the wind at Station X is reported to be *K* knots. I was told that the reported speed is a l-minute average. The observer is supposed to watch the wind speed dial for 60 seconds and

Listing l-continued

```
        JMP MID100              "
        JMP MID100              "
        JMP  TB100              ;Top and bottom of '100'
;
TB100   SETB  SIG               ;Top and bottom of '100'        [X_XXX_XXX_]
        CLRB  SIG
TB50    SETB  SIG               ;Top and bottom of '50'         [XXX_XXX_]
        JMP   $+1               ;(Two-cycle NOP.)
        CLRB  SIG
TBO     SETB  SIG               ;Top and bottom of '0'          [XXX_]
        JMP   $+1               ;(Two-cycle NOP.)
        CLRB  SIG
        RET

iM50    SETB  SIG               ;Top middle of '50'             [X___X_X_]
        CLRB  SIG
        JMP   MIDO
MID50   SETB  SIG               ;Middle of '50'                 [XXX_X_X_]
        JMP  001010
BM50    JMP  0101010            ;Bottom middle of '50'          [__X_X_X_]
;
MID100 SETB  SIG                ;Middle of '100'                [X_X_X_X_X_]
        CLRB  SIG
        SETB  SIG
        CLRB  SIG
D101010 SETB  SIG                                               [ X_X_X_]
 001010  CLRB  SIG                                              [_X_X_]
MIDO    SETB  SIG               ;Middle of '0'                  [X_X_]
        CLRB  SIG
        SETB  SIG
        CLRB  SIG
        RET


.==================================================================

; Section 1f: Example of Foreground Routine Cooperative Multitasking

        MOV   REP,#65
LOOP    CALL  HSYNC             ;Keep the appointment
        CALL  MAIN              ;Continue with main task
        MOV NEXT,W              ;Save continuation pointer in NEXT
        DJNZ  REP,LOOP          ;Loop back for another HSYNC


;
MAIN    MOV  W,NEXT             ;Get the saved pointer into W
        JMP  PC+W               ;Add W to the program counter
;
        JMP  MAINO              ;This is just a jump table indexed by W
        JMP  MAIN1
        JMP  MAIN2


        .
        JMP   MAINn
;
MAINO   ..                      ;Do a little bit.
        RETW  1                 ;Return to MAIN1 on next call
MAIN1   ..                      ;Do a little bit more.
        RETW  2                 ;Return to MAIN2 on next call
MAIN2   ..                      ;Do a little more yet
        TEST  VALUE             ;Is VALUE zero?
        SNZ                     ;(Skip if nonzero)
        RETW  0                 ; Yes: Return to MAINO on next call
        RETW  3                 ; No: Return to MAIN3 on next call
MAIN3   ..                      ;Do some more
        RETW  0                 ;Loop back to MAINO on next call
```

estimate the average velocity during that interval, accounting for gusts and lulls. In fact, after many years watching official observers at work, the consultant had never seen this procedure actually used. A quick glance at the dial was what passed for an official observation. Perhaps technology can make up for human laziness after all.

After each sample, the values in NPROP and EPROP are used to calculate the instantaneous speed. The difference between this speed and the current dial speed is saved in another register in sign/magnitude format. This data is for calculating the smoothly changing dial pointer position that the screen displays. The routine that performs these calculations is called during video line 195, just before the dial pointer is drawn. Thus, the dial position is updated once each frame, even though the props are sampled only once every 64/60 of a second. Delayed by 1 second, the smoothly animated visual effect is much more pleasing than 1 -second jumps of the pointer.

The gust speed is derived from the dial speed. The dial speed is compared with an exponentially decaying variable **(G**UST) corresponding to a previous gust speed. If the dial speed is greater than or equal to GUST, both GUST and the displayed gust speed are updated to the new, higher value. The decaying variable is then allowed to fall about once a minute using the formula

```
GUST := GUST-1-INT(GUST/16)
```

With no wind, it will eventually reach zero, and the displayed gust speed will also revert to zero at this time. This feature prevents the display from getting stuck on a high gust value. Unlike the average speed, the gust speed can change while it is being displayed.

That's about all there is to it. The video signal constantly refreshes in a continuously executing loop, and the wind speed calculations are carried out in small bits during the first group of blank lines. The PIC, though primitive and cheap, is fast enough to do it all! But not in color.. .yet.

FUTURE DIRECTIONS

Many more things could be done with these techniques. A temperature sensor added to the wind gauge would allow not only a display of outside temperature, but also a calculation of windchill. A little external RAM might afford a display of wind history over the last 24 hours. In other areas, how about a video voltmeter or a host of other cheap video instruments? With Microchip Technology having just announced their PIC 16C71, complete with a multiplexed A/D converter, an eight-level subroutine stack, and more registers, still in an 18 -pin package, the possibilities seem endless. ❑

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.

Philip Pilgrim owns and operates Discovery Bay Software Co., an electronic hardware and graphic arts ***design firm.***

REFERENCES

PIC16C5X ***Development Tools,*** Parallax Inc. (Citrus Heights), 1991.

PIC16C5X ***Reference Manual,*** Microchip Technology (Chandler), 1991.

Horowitz, Paul, and Winfield Hill, ***The Art of Electronics,*** Cambridge University Press (Cambridge), 1980.

Liff, Alvin A., ***Color and*** *Black-****and-White Television Theory and Servicing,*** Prentice-Hall (Englewood Cliffs), 1979.

IRS

401 Very Useful
402 Moderately Useful
403 Not Useful

Computer-controlled Light Dimmers

Part 1: Theory

FEATURE ARTICLE

William Von Novak

The use of computers to control theater lights has grown by leaps and bounds in recent years. In this first part, William explains the theory behind stage lighting and electronic dimmers.

Anyone who has ever watched TV, gone to a movie, or seen a live show has seen theatrical dimmers in action. Everyone from the smallest rock band to Steven Spielberg has used lighting controls to light a scene, create a mood, or enhance the action. This two-part article will discuss some issues in designing and building dimmers for theatrical lighting. I will end with a design example.

Stage lighting has come a long way from the first DC arc lamps. The original requirement for theater lighting-allowing the actors to be seen by the audience at night-has been expanded tremendously. Modern lighting designers (LDs) want to control the color, intensity, focus, motion, and pattern from their lighting instruments. These new requirements have given rise to an incredible number of specialized instruments, each with its place in an LD's "toolbox." The instruments we will concentrate on are the old standbys—the 500- to 2000-watt incandescent leikos and fresnels. Leikos give a narrow, controllable beam of light, while fresnels give a fuzzy "wash." These venerable fixtures still make up 80–90% of the instruments in a typical small-theater lighting system.

The first dimmers were resistance-type. A variable series resistor allowed the LD to vary the intensity of the lights. Although simple, they dissipated tremendous amounts of power (up to half of the rated load) and the wattage of the load had to be held constant. "Phantom Lamps" (instruments that didn't emit usable light) were often hung in corners of the stage to fill a dimmer's wattage requirement. The resistance dimmers were, however, well suited to the 100-volt DC systems of the time.

One early variation on the resistance dimmer was the bucket dimmer. A steel bucket was filled with salt water and an electrode was slowly lowered into the water. Full on occurred when the rod touched the bottom of the bucket. A fan blew the resulting gases away. This does work—I've tried it-but it has one or two practical drawbacks.

As AC began to gain popularity, most theaters switched to variac dimmers. The variac transformers allowed direct control of voltage rather than resistance, so the load could vary up to the transformer's maximum rating. They were efficient but huge (a 1-kW dimmer could weigh 40 pounds) and had to be controlled at the dimmer itself. This handicapped the operator, who usually could not see the result of an input from the backstage dimmer panel. Motor drives helped somewhat but had their own set of problems.

The first electronic dimmers used thyratron tubes to do a kind of phase control on the AC. The tubes were big, bulky, expensive, and fragile, and the control circuits were complex. Not many theaters could afford the racks of equipment necessary or put up with the noise of the fans they required.

With the advent of SCRs, phase control became practical. These dimmers allowed the control panel to be placed far from the dimmer panel, usually somewhere on the balcony where the operator has a clear view of the stage. Their high efficiency meant smaller packages and lower cost. Suddenly LDs found themselves with enough dimming channels to do all those neat effects they'd always wanted to try. The introduction of the phase-control dimmer marked the beginning of modern theatrical lighting.

A typical modern lighting system for a small theater consists of three main components: the dimmer packs, the dimmer control board, and the instruments themselves. The dimmer packs are usually hidden away near the power source, and the instruments are

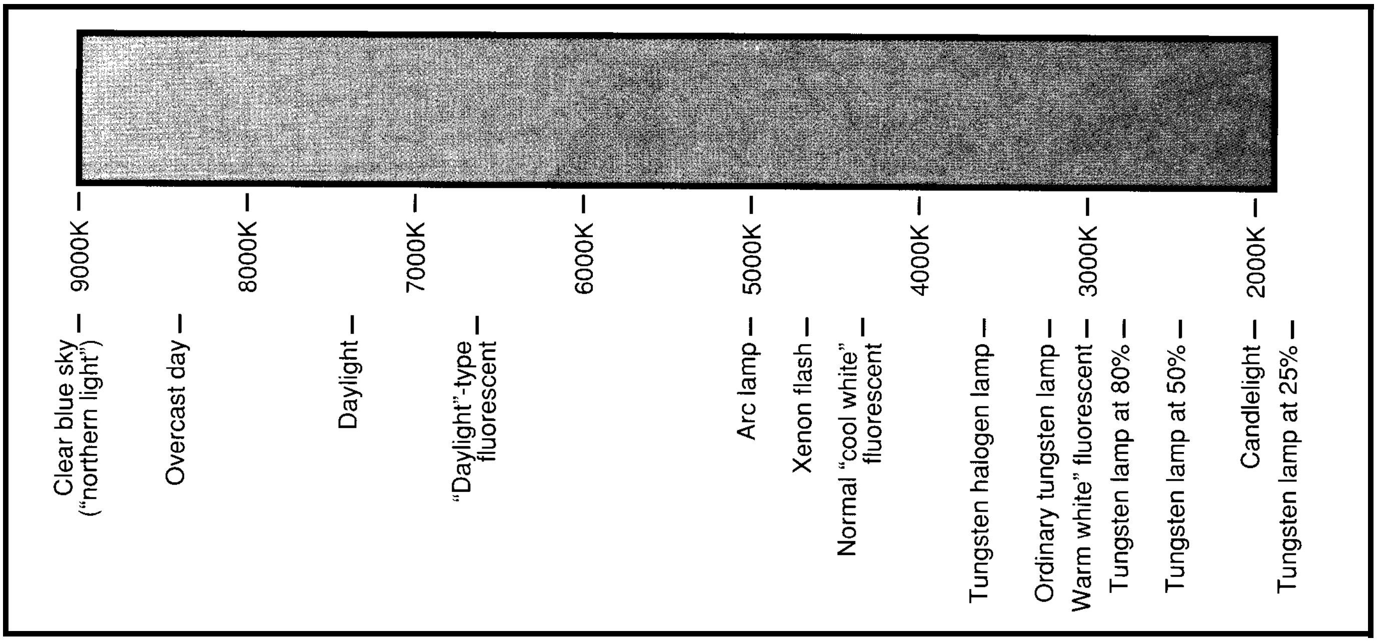


Figure 1—*Color temperature is used by lighting designers to measure how "cool" or "warm" light is. A c/ear blue sky and candlelight represent the two extremes. All temperatures are in kelvin.*

cabled to the dimmers directly. The dimmers receive low-level control signals from the control board, which is usually located some distance from the backstage. Most modern boards have several presets, or "scenes." Each preset is a stored value for every dimmer channel that can be recalled easily. A common simple arrangement is the A/B preset, where the current dimming levels are set on the "A" set of slider controls and the next scene is set up on the "B" set. A master lever is used to fade between the two.

The control signals running from the board to the dimmers can take several forms. The oldest and still most popular is the 0-10-V scheme, where 10 volts represents full brightness and each dimmer has its own wire. This arrangement has some obvious problems for 400-dimmer systems. There have been several analog-multiplex schemes, where 200 or so analog levels are multiplexed onto one wire along with control and framing information.

A new digital standard, DMX5 12, has recently emerged. DMX5 12 multiplexes 5 12 channels of dimming, each with 256 possible levels, onto one pair of wires. It uses a twisted pair to send a 250-kHz RS-485 serial data stream to the instruments. Each instrument has a unique address on the bus. (Some instruments, like motorized effects, have several sequential addresses for intensity, x position, y position, color, etc.) DMX5 12 looks like the most promising universal replacement for the old 0-10-V system.

INCANDESCENT LAMP PHYSICS

When designing a dimming system, the most logical place to start is with the controlled device: the lamp. A typical lamp takes power and uses it to heat up a filament to several thousand degrees. In the process, about 10% of the energy is converted to visible light. (That's one reason environmentalists are pushing for compact fluorescent lights for the home.) This light is focused by the instrument and illuminates the stage.

Seems simple. If the lamp were always operated at 100% power, it would be. But since LDs insist on turning lights on and off and dimming them up and down, their dynamic behavior becomes very important.

When a lamp is first turned on, the resistance of the cold filament can be 20 times lower than its warm resistance. This characteristic is good in terms of quick warmup times, but means that 20 times the steady-state current will be drawn for the first few milliseconds of operation. The semiconductors, wiring, and fusing of the dimmer must be designed with this inrush current in mind. Also, since the filament has a finite mass, it takes some time (depending on lamp size) to reach operating temperature. This delay is perceived as a "lag," and limits how quickly an effect can be dimmed up.

One way to deal with both of these problems is to preheat. Preheat keeps the filaments hot but not incandescent, thus reducing turn-on surge and lag.

The ideal lamp would produce at 50% light output with a 50% power input. Unfortunately, incandescents aren't even close. Most require at least 15% power to come on at all, and afterwards increase in intensity at an exponential rate. To make it even more complicated, the human eye perceives light intensity as a sort of inverse-log curve. To get around these problems, most dimmer manufacturers incorporate intensity curves in their control circuits in an attempt to make selected intensity more closely approximate perceived intensity. Most resemble sideways S curves, and many are proprietary.

A final issue in lamp physics is color temperature-how "cool" or "warm" the light produced by a lamp is, measured in Kelvin. A cooler light has a higher color temperature and looks more bluish. A candle is about as warm as you can get-around 2000 K

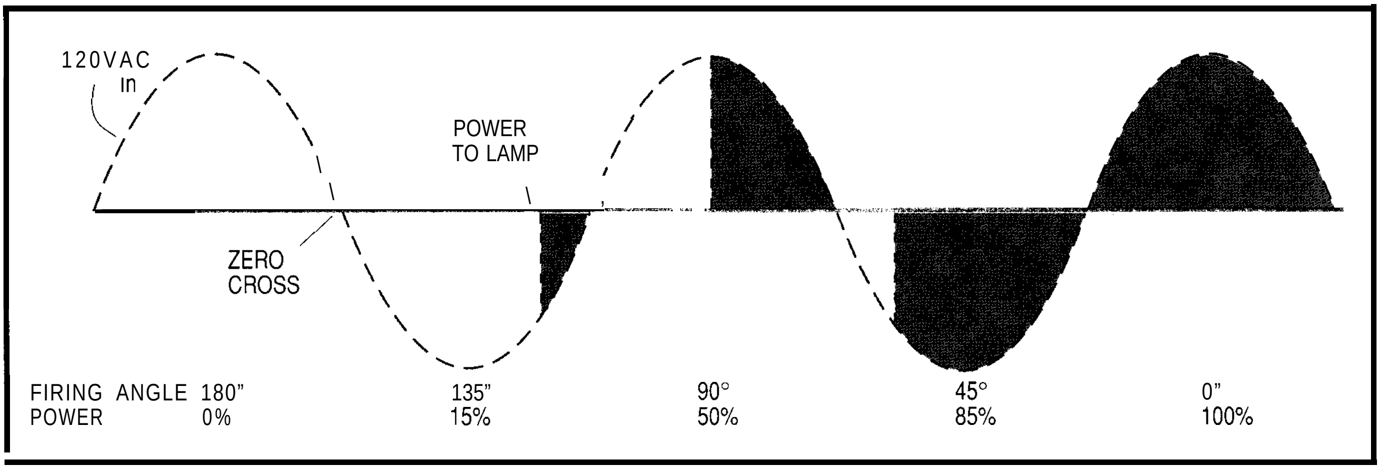
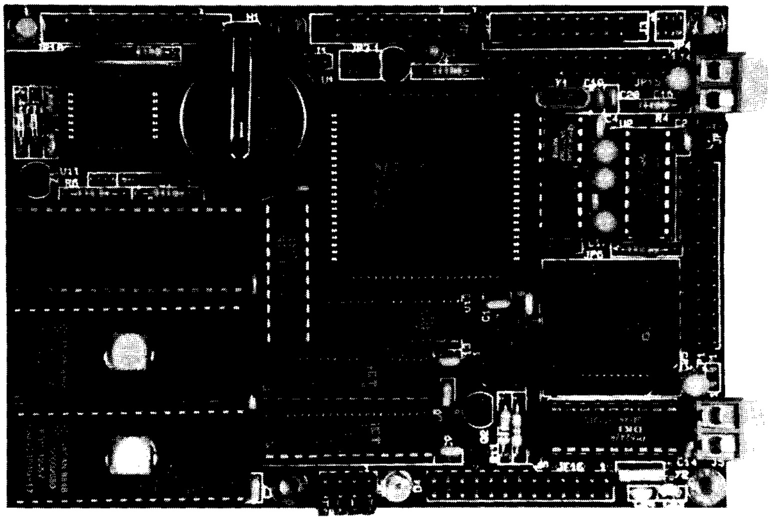


Figure 2—*To dim an incandescent light, the /amp is turned on by the dimming circuit some time after zero crossing and automatically turns off at the next zero crossing. The final intensify is determined by the shaded area under the curve.*

(see Figure 1). An overcast day is about as cool as you can normally get at around 8000 K. Most incandescent lamps operate around 3000 K. LDs use gels (colored, plastic-like sheets) to change the lamp's color and maximum brightness. This technique usually works fine until the lamp is dimmed. As the filament cools off, the color temperature drops (which makes sense) and the light begins to look warmer and more reddish. The midnight-blue scene that looked so good at 75% power suddenly becomes purplish and muddy as the instruments are dimmed and the color temperature drops toward zero. There's no easy way around this color shift with conventional instruments. LDs must plan for the color shift and use a gel that gives acceptable color at both ends of the dimming range.

THEORY OF OPERATION

The basic strategy behind the phase-control dimmer is to turn the lamp on and off quickly enough that the thermal inertia of the filament “smooths out” the resultant power level. This is done by waiting some time after the AC line zero crossing to turn the lamp on, then turning off at the next zero crossing. If the delay is 4.16 milliseconds, or half-way between



zero crossings, the lamp is at 50% power. From there, an area-under-the-curve formula is necessary to determine what firing delay gives what power level (see Figure 2).

The heart of any modern dimmer is the semiconductor. In most cases the switching element is a pair of SCRs connected in inverse-parallel. SCRs are ideal for dimming due to their turn-on characteristics. It takes only a few milliamps at their gate terminals to turn them on, after which they latch on until the current through the main terminals goes to zero. The result is a simple drive circuit: turn on by a low-power pulse and turn off automatically at the end of the AC cycle.

One useful variation on the SCR-type device is the triac. A triac is basically two SCRs on one piece of silicon, allowing one device to replace the two SCRs.Triacs do work, but sometimes won't turn off with highly inductive loads. A few experiments have been done with GTOs (gate turn-off devices), but for dimmer applications they don't have significant advantages over SCRs.FETs and IGBTs have been used with some success. They have the unique advantage of being able to operate linearly to reduce current rise times.

Often, the dimmer circuitry must be isolated from the power supply for safety or wiring code reasons. The SCR itself cannot be isolated because it switches the 120-volt power directly. Isolation is most often done at the next closest place to the power line: the SCR trigger circuit. The two most popular ways of achieving isolation are with pulse transformers and opto-couplers. Pulse transformers couple an AC pulse to the gate of an SCR. Since the SCR will latch on after it is triggered, a sustaining DC level is not required. Pulse transformers are cheap, rugged, and can be tailored to SCRs with unusual drive requirements.

Using optocouplers to provide isolation was made much easier by the introduction of the MOC3010 series of optocouplers. The MOC3010 is an LED and a light-sensitive triac mounted in a common package. The triac is connected between the gate and MT1 terminals of the power triac. When the LED is lit by the dimmer circuit, the small triac turns on, firing the power triac. Since the small triac has the same turn-on and turn-off characteristics as the larger triac, control of the device is easy.

For super-cheap applications, isolation isn't necessary. Typical hardware store wall-mounted dimmers are nonisolated, and for theatrical dimmers the isolation can be done at the other side, near the control inputs. It isn't easy, though, to pass a 0-10-V signal across an isolation barrier.

Next month, I'll discuss filtering, snubbers, and the design of the dimmer project itself. ■

William Von Novak holds a B.S. in electrical engineering from MIT and is head of R&D at Custom Power Systems. His specialties include dimmer systems, fluorescent ballasts, motor control, laboratory data acquisition, and audio DSP.

I R S

404 Very Useful
405 Moderately Useful
406 Not Useful

DESIGN CONTEST WINNER

FEATURE ARTICLE

Control Your Telescope by Voice

Michael Swartzendruber

With just six voice commands (eight with a little more work) your telescope will be doing tasks that once left you fumbling in the dark under the night sky. This project might be a good excuse to give amateur astronomy a try.

Amateur astronomers are a nocturnal bunch. The very nature of their hobby demands it. Good, deep-sky viewing requires darkness, but that can also be a bit of a hindrance, especially when you want to adjust the position of your telescope. Over time, astronomers develop a sense for the location of the declination and right ascension fine adjustments on their instruments to compensate for working in the darkness. This sense comes from many hours spent groping in the darkness for these controls; a solution that is workable, but sometimes distracting.

Telescope-manufacturing companies have developed and marketed a number of devices that position the telescope, including motors driven by simple switch boxes or computers with built-in "star maps." These methods are nice tools for semiautomated telescope positioning, but they also require hand operation in a low-light-level viewing environment. However, there is another option that does not require the use of the hands.

Imagine what it would be like to have the telescope change its position with a spoken command. For instance, saying "Go" would cause the declination control to raise the angle of the telescope while saying "Reverse" would lower the angle. Sound like science fiction? Well, it's not. The technology for this feat is well within the budget of the amateur astronomer. In fact, the electronics for this system cost about $75. I'd like to give you the opportunity to build a system I've named the Scope Commander. It uses the VCP200, an economical, speaker-independent, word-recognition chip from Voice Control Products Inc. This remarkable little chip represents the start of a new era of voice-commanded appliances and control systems.

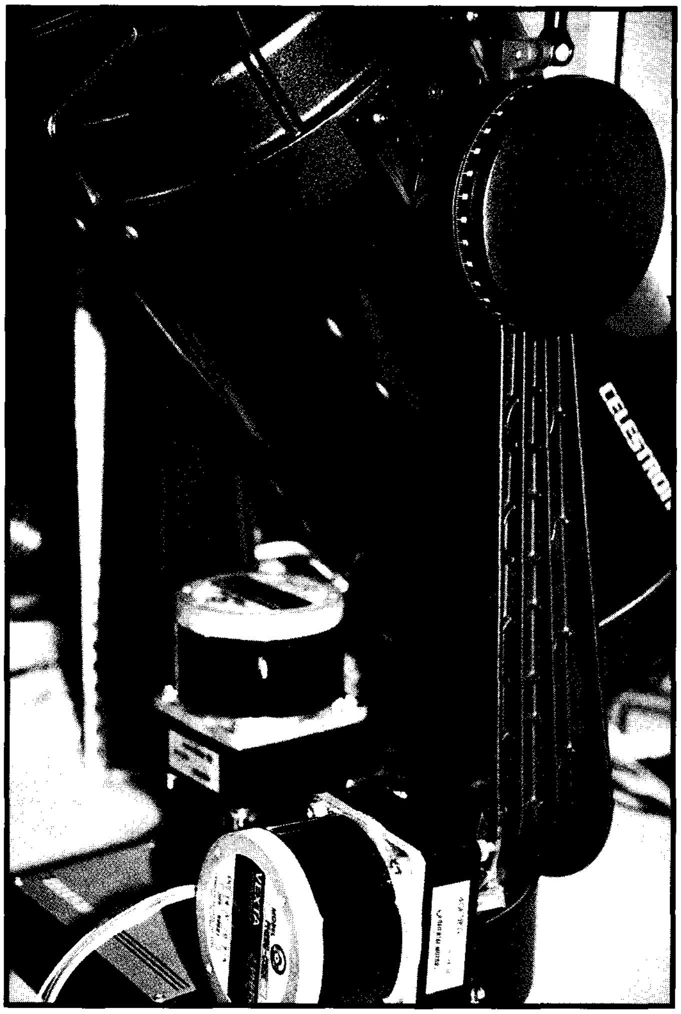


Photo I--The Scope Commander can be used with many telescope mounting schemes. *The prototype is used to control a Celestron C8 telescope.*

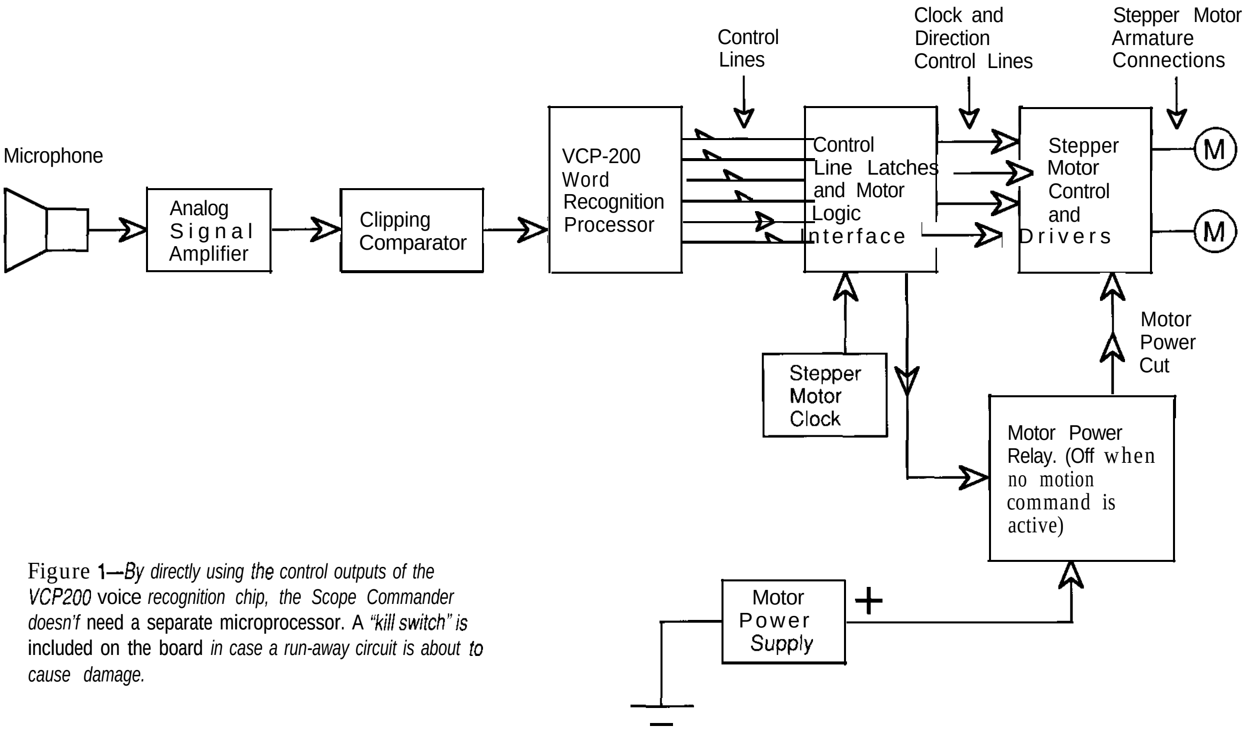

Microphone
Analog Signal Amplifier
Clipping Comparator
VCP-200 Word Recognition Processor
Control Lines
Control Line Latches and Motor Logic Interface
Clock and Direction Control Lines
Stepper Motor Control and Drivers
Stepper Motor Armature Connections
M
M
Stepper Motor Clock
Motor Power Cut
Motor Power Relay. (Off when no motion command is active)
Motor Power Supply
+
–

Figure 1—*By directly using the control outputs of the VCP200* voice *recognition chip, the Scope Commander doesn't* need a separate microprocessor. A *"kill switch" is* included on the board *in case a run-away circuit is about to cause damage.*

Experimentation with voice recognition is an ongoing pursuit for many computer specialists. Several word-recognition systems have been developed for experimenters, professionals, scientists, and hobbyists over the years. Some of these systems consist of peripheral cards and software that you must add to a PC. The first system I can remember was used with S-100 bus systems. Macintosh and AT-architecture platforms use more current designs, but who wants to lug around one of these machines with a telescope? I wouldn't and that's why Scope Commander requires a much smaller, stand-alone system.

In a brief overview, word-recognition systems work by accepting the sounds of the human voice as input. For examination purposes, the words are converted to either a succession of electrical waves, a pattern of waves that represent the phonemes contained in the spoken word, or both. A *phoneme* is defined as a sound created within the vocal and nasal cavities of a speaker that is modified by the size and shape of the mouth's opening and by the simultaneous positions of the tongue and teeth. A distinct succession of phonemes is recognized as a spoken word.

The phoneme patterns of a particular word are represented to a computer system as a distinct frequency mix that changes over time Then the system converts the phonemes to frequency patterns and compares them with a previously stored representation of frequency patterns held in a memory array. It declares a match when the patterns contained in the input data stream match the stored patterns precisely enough.

The VCP200 chip is a Motorola 6804 microprocessor preprogrammed to create the core of a word-recognition voice signal-processing system. This core simplifies experiments in word recognition because the ROM on the chip contains the word-recognition algorithm, so the experimenter does not have to develop it. Also, the $15 price makes the device affordable. The processor's preprogramming limits the system's eventual range of capabilities in a gross sense, possibly keeping the VCP200 out of some complex systems, but it is ideally suited for the purposes of small, dedicated, voice-controlled systems like the Scope Commander.

The computer hardware requirements of the VCP200 are minimal. It needs little more than a crystal and a S-volt supply to operate, making this system lightweight, portable, and usable in many dedicated systems. Other circuit blocks in the Scope Commander perform voice input signal conditioning, output data latching, and steering logic and load current control and switching.

As shown in Figure 1, the system consists of a voice signal input and analog signal-conditioning circuit; the VCP200 processor circuit core; and the latching motor control interface, which includes a stepper motor clock source and high-current motor drivers.

CIRCUIT DESCRIPTION

The voice signal input and conditioning module is composed of an electret microphone, a microphone preamplifier, a final amplifier stage, and a clipping comparator. The microphone preamp amplifies the very low voltage output of the polarized electret microphone. This signal is passed to a final amplifier stage.

Adjust the gain of this stage to compensate for the volume of your voice by modifying the value of the feedback resistor connected between pins 6 and 13 of the op-amp. The output of the final amplifier is sent to a clipping comparator, which jumps to positive rail when the voice signal amplitude exceeds a certain preset comparator amplitude.

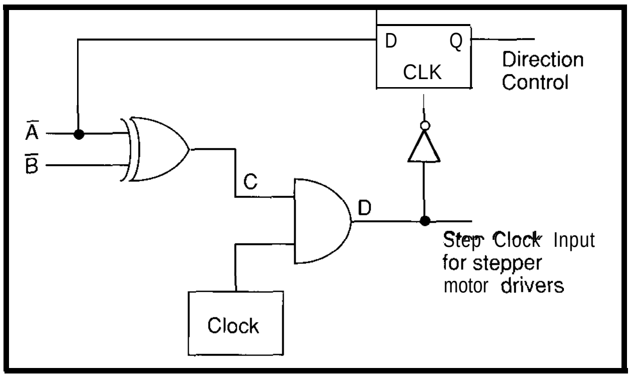


Figure 2—*When either* input *signal (A or B) is active, fhe clock signal is gafed through to the latch. The latch uses the A input to determine the final state of the motor direction line.*

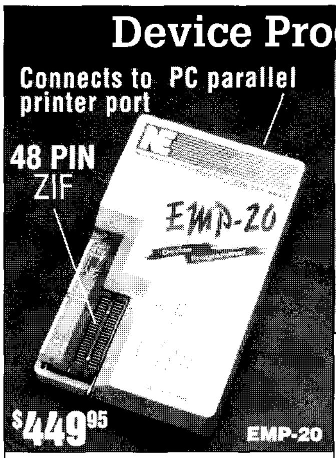
If you need a variable-sensitivity circuit, you can make the threshold level adjustable. Use the sensitivity control to make the circuit suitable for environments with different levels of ambient noise conditions. Adjust the threshold by replacing the 5.6-kΩ resistor (connected between pins 5 and 12 of the op-amp) in the comparator circuit with a 10-kΩ pot whose wiper connects to the positive input of the comparator (pin 12). You must also substitute a 1-kΩ resistor in place of the 4. 7-kΩ resistor (connected between pin 12 and ground) that is a part of the bias network of the comparator circuit to complete the modifications.

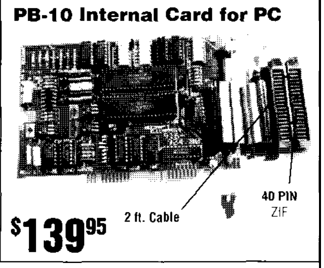
Note the approximate total of these two resistances is 11 kΩ. This amount is necessary to maintain the 2.5-volt bias on the positive inputs of the preamplifier and the final amplifier stages that precede the comparator stage. The bias tricks these circuits into behaving as if a ±2.5-volt supply was powering them. As long as the total value of the threshold circuit resistors is 11 kΩ, the other op-amp circuits will be correctly biased and will operate properly. The larger the value of the potentiometer, the more range the threshold circuit will have.

The frequency at which the voice signal crosses the comparator threshold voltage is closely akin to the frequency of the sound waves the speaker's voice produces. This succession of frequencies relates to the series of phonemes (or words) spoken. Thus, the threshold switching action of the comparator modulates the frequency of the human voice as a digital logic level pseudo pulse train. The frequency-modulated signal train represents a spoken word.

The VCP200 processes this digitized signal to match an internally stored pattern. The exact processes of the algorithm remain the property of Voice Control Products Inc., so I could only speculate about what they have written in the program stored in hard ROM code in the 6804.

When the VCP200 determines a pattern match, it toggles one of its eight output lines. The programming in the VCP200 is set to pattern match on the words *Lights, Go, Slow, Reverse, Left* Turn, *Turn Right, Stop,*





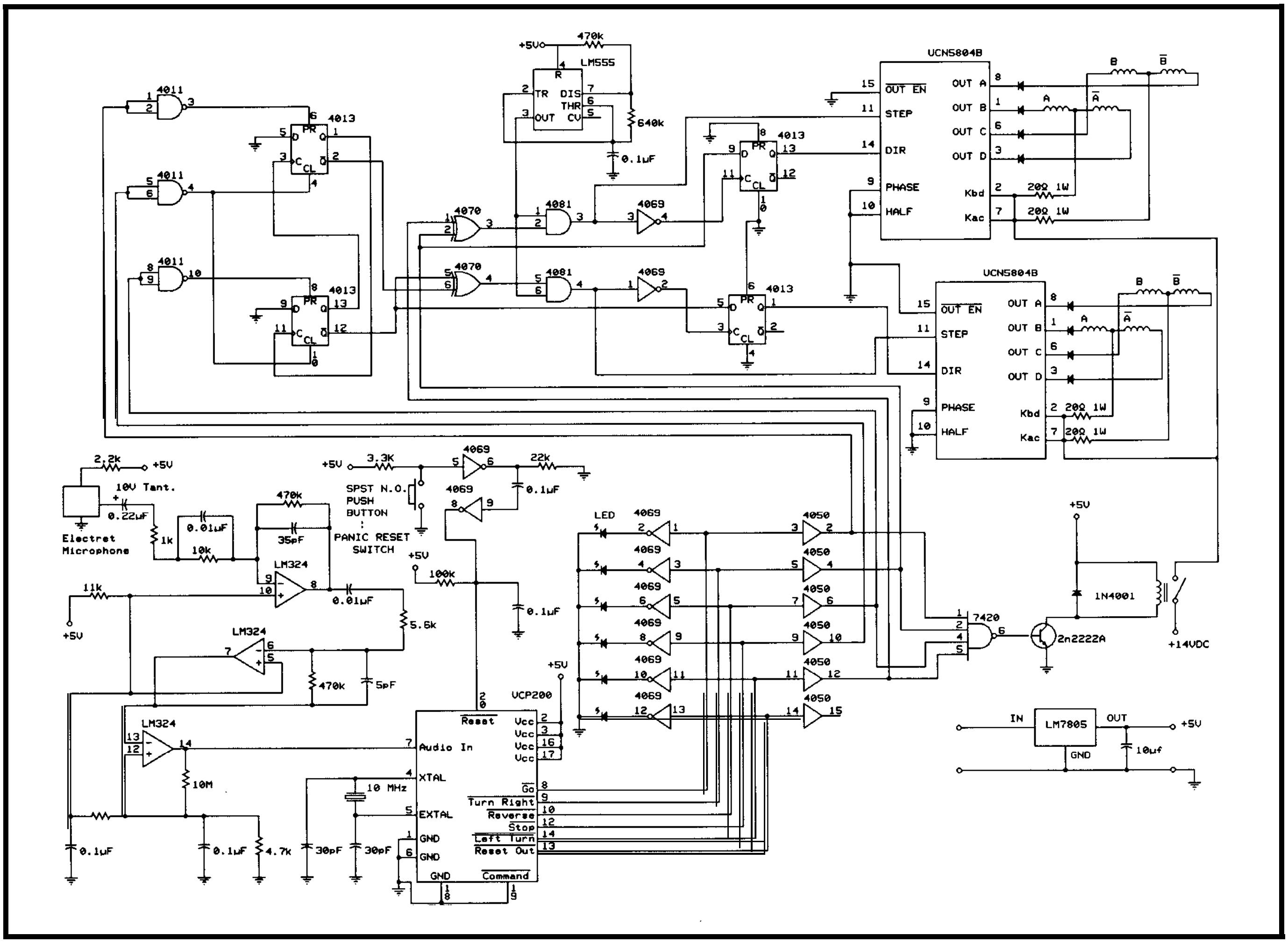


Figure 3—The heart of the Scope Commander is the VCP200 voice recognition chip. The UCN5804 fakes care of the details of driving the stepper motors. With the addition of some gates and latches, the circuit eliminates the need for a separate microprocessor.

and *Reset.* The Scope Commander project uses Go, *Reverse, Left Turn, Turn Right, Stop,* and *Reset. Left Turn* and *Turn Right* are for right ascension fine adjustment. Go and *Reverse* set declination. *Stop* halts all motor rotation. The *Reset* line is buffered, but it is not implemented here.

The other two commands *(Lights* and Slow) are also not implemented here, but you may use them for focus control by duplicating one of the motor steering logic sections. The VCP200 output lines used in this project are sent to two gates. The first is an inverter that drives an LED indicator and shows which control line is low at the time. This feature is useful because sometimes the circuit will mistakenly trigger or misunderstand a speaker's intention. The second gate is a noninverting buffer and driver chip, which isolates the VCP200 from the rest of the circuit and gives the "bus" a little more current capacity.

The declination control lines (Go and *Reverse)* are inverted, so they can be active high, and are latched into a pair of Set/Reset latches. *Stop* resets these latches. This latching circuit lets the right ascension control bits *(Turn Right* and *Left Turn)* change states, while the state of the declination control bits (Go and *Reverse)* remain unchanged.

The latched circuit has three basic steering modes to the Scope Commander circuit: adjust right ascension, adjust declination, and simultaneous adjustment of both ascension and declination. These steering modes can be made independently of either motor's direction. Use Stop between commands to lessen the chance of mechanical binding or other anomalous behavior in the Scope Commander circuit. The circuit will change states without a hitch, but rapidly changing the stepper motor states could damage the motor armature. I included the reset switch as a safeguard to allow an emergency system halt.

The stepper motor interface is a straightforward application of a pair of "L/R" stepper motor driver chips and a 555 timer. The trusty old 555 is set up as a bistable multivibrator running at 10 Hz. You may use a higher rate if necessary as long as you stay within the limits of the drivers.

The pulse train from the 555 is used as the step clock for the stepper motor driver chip. The UCN5804 from Allegro Micro Devices (formerly sourced by Sprague Semiconductor) contains logic that generates the proper pulse sequences to drive the stepper motors. The 555 frequency output is routed through the AND gates in the motor logic steering section and applied to the logic clock input on the UCN5804.

Each of the motor direction control bits from the VCP200 is passed through an XOR gate. This gate will go high when one of the control bits connected to it goes low. Only one of these control lines can go low at a time when the VCP200 determines a word match. When both of its inputs are high, the XOR gate is in its “inactive” condition, and the word associated with that control bit has not been recognized.

Once a word is recognized, the VCP200 pulls the control bit associated with that word low, then the XOR gate is “activated” and goes to a high state. The high state goes to the input of an AND gate whose output then follows the toggle clock connected at its other input, passing the clock signal to the L/R driver chip clock input, which converts each incoming pulse to the next valid step state for the motor armature.

The output of the AND gate is also routed through an inverter, and this signal is used as the clock source for a *D* latch (see Figure 2). Its data bit connects to one of the channel’s control bits. The output of the *D* latch changes state on the falling edge of the clock. The Q output from this gate sets the direction control line on the stepper motor driver chip. Because only one of these lines will be low, the Q output of this gate will be low or high depending on the state of its *D* line to activate the XOR gate, giving direction control to the UCN5804 chip. The inversion of the clock signal exiting the AND gate is required for properly changing the state of the L/R signal in relation to the clock’s transition state.

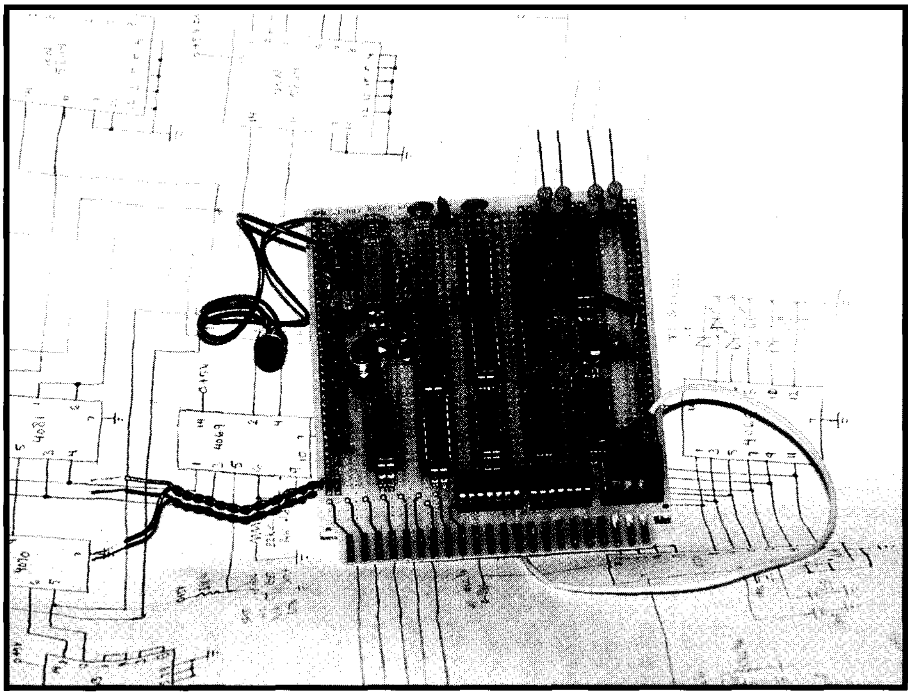


Photo 2—*The* final prototype is *built on protoboard* using *wire-wrap construction.*

The output of the UCN5804 operates up to 24 V and 1.5 A peak. This device has a maximum clock input of 320 Hz, so the clock speed of 10 Hz is well within its specification. The armature resistors on pins 2 and 7 are used as short circuit and current protection. The Schottky diodes, in series with the stepper motor driver chip’s bipolar output devices and the stepper motor armature, are implemented to control back EMF from the collapsing fields of armature poles.

The stepper motors are 74-oz/in, four-phase (or two-phase center-tapped) unipolar steppers with a step angle of 1.8” per step. These motors require 14 volts and have a peak coil winding of 0.7 A. This circuit operates at 0.5 A while the motors are running, so they function well within their tolerance ratings. The motors are connected to the telescope adjustment controls through 9: 1 reduction gear boxes. The output shaft of the gear box directly drives the fine position adjustments on the optics system. The rotational velocity of the motor shaft is 18" per second because these motors are stepped at 10 pps. The end shaft on the gear box is rotating at 2" per second after gear reduction.

The motor windings do not have to be charged while the motor is not being stepped because the gear box provides a significant amount of holding torque. Therefore, you can shut off the armature current when the motors are not being moved. This feature keeps the motor's temperature down and lowers the constant current requirements of the system, an important factor because the circuit will usually be run on a battery-powered supply.

A quad input TTL NAND gate, whose output drives the base of a 2N2222 (or equivalent) transistor, controls the current for the stepper circuit. This transistor is wired as the classic grounded emitter switch. The current in the collector circuit drives a relay. A diode wired in the coil circuit protects the collector of the transistor from back-EMF fields. The output of the NAND gate will go high when any of the motor control lines (Go, *Reverse, Left Turn, Turn Right) goes* low. This change will close the relay contacts to power the stepper motor as long as it is being driven.

The system requires 5 VDC for the main logic board and from 12 to 14 VDC for the stepper motors. A car battery and a regulator should do quite nicely. Be sure to take care when transporting the battery to prevent fracturing the case or spilling acids; a carrying box with a handle might not be a bad idea. I also recommend equipping the system with a reverse voltage protection circuit to prevent backward battery mishaps.

MOTOR-MOUNTING CONSIDERATIONS

For me to address all the issues that could arise when mounting motors to the device is impossible. Many different scopes are available and they may have slightly different locations for declination and ascension adjustments, but you should follow these guidelines.

Take care when mounting the motor and gearhead assemblies to the telescope; don't just start drilling holes everywhere to mount the motors. Place the motor shafts and the fine adjust control shafts on a straight line to prevent or minimize mechanical binding. Mount the motors in a way that does not impair operation or storage of the telescope. Finally, the motors should be easily removed if you need to operate the telescope without them.

CONCLUSIONS

Scope Commander is a useful application for voice control technology. The thrill you'll feel when your device responds to a spoken command is well worth the time and effort you will have invested. This project is relatively inexpensive to build; the components for the logic assembly I described cost about $70 retail.

The Scope Commander brings a new level of enjoyment to deep-sky viewing. Finally, as a word of caution, don't let the scope ocular bump you in the eye as it is moving. I wish those of you who build this project many happy hours of star gazing. Remember, if there is a limit to the number of possibilities the universe has to offer, someone will probably find it in deep space. ❑

Michael Swartzendruber is an engineer with experience in network and communications design and Windows and Macintosh programming.

SOURCE

Voice Control Products, Inc.
(408) 647-1502

I R S

407 Very Useful
408 Moderately Useful
409 Not Useful

Add Interrupt Support to Polled Parallel Ports

FEATURE ARTICLE

James Grundell

Having to constantly poll a parallel port to watch for changed input bits can eat up a large portion of a processor's time. With the right kind of parallel port, a few extra chips can ease the processor's burden sigificantly.

When designing control electronics for embedded applications, determining the best approach to provide access to the interrupt structure and I/O facilities could be likened to a game of pin the tail on the donkey, more so if the board is targeted for a broad market. A parallel interface chip could be used, or a bus-based system could be constructed. The interrupt lines and the data bus could be brought out to a connector or a dedicated interrupt controller might be employed. I'm sure there are several other methods that I could mention, but you get the idea.

What manufacturers are providing as a workable I/O system in conjunction with the interrupt structure is as varied as the number of SBCs or embedded system solutions currently being offered. Considerations such as cost and time, and compatability with existing software and tools enter the picture and begin to influence the decision of whether designing a custom controller or purchasing an off-the-shelf solution is best. Such was the case on a project I was involved with some time ago.

I ended up selecting an off-the-shelf board based on the Hitachi HD64180 for the following reasons: It provided a number of solutions to the issues discussed above all wrapped up in a small compact board and it provided downward compatibility with many custom control systems in use in the field right now based on the 8085 and 280. Many of the older development tools (with macro and conditional assembly features) and even some simulators designed for the 8085 or Z80 can still be used when in a pinch. The expansion bus used on the board also provided an added incentive for selection.

One thing that is missing from the board I selected, as well as many other single board computers, is a built-in parallel I/O interrupt system. I can understand why the designers of these boards might have neglected this circuit support in the past (board real estate and parts count), but as these boards continue to take on applications once reserved for custom or larger dedicated systems, the inclusion of these smaller support circuits can impact buying decisions greatly. Given a choice between an interrupt-driven system or a polled one, I would opt for the interrupt system any way I could get it. The event handling is cleaner and much more well defined, which leaves more time to deal with the core of the problem.

You could just poll the port on a regular basis, and most of the time it will work just fine. However, there are times when the application is using all the resources of an inexpensive SBC or embedded controller, cost is a major factor, or the time frame in which to complete the project is weeks instead of months. For example, the controller may have to monitor two serial ports, an ADC, and perhaps two dozen digital inputs. Polling all that I/O quickly leads to spaghetti code, sluggish performance, and missed data. The project can't afford to dump money into a dedicated PLC let alone the

Figure l--The 8255 *PPI* supports three different modes for ifs three B-bit *I/O* pork. *By using the control lines on port C and a handful of parts, you can* generate a processor interrupt any *time one of the inputs bits on port A changes.*

Port	Mode 0	Mode 1	Mode 2
A	Basic I/O	Strobed I/O	Bidirectional Strobed I/O
B	Basic I/O	Strobed I/O	N/A
C	Basic I/O	Control/Status for Ports A & B	Control. Status for Port A

development tools. The PLC route probably doesn't offer the additional features required for the system anyway. This is a situation where the selection of a flexible controller design and a little custom circuitry pays off.

The board I selected uses an 8255 Programmable Peripheral Interface (PPI). As the name implies, the PPI allows a varity of I/O combinations or "modes" to be programmed. One of these modes provides interrupt-driven I/O, which is what I'm after. By adding three ICs and few connectors, an interrupt generator can be constructed for an S-bit parallel port.

THE HARDWARE

The main IC used to detect a change in state in any of the inputs is a 74LS280—a 9-bit odd/even parity generator/checker. The ninth bit is used to "reset" the chip after detecting the change. In order to effect the reset, both halves of a 74LS74 dual D-type flip-flop are used to provide an alternating level to the ninth bit input of the LS280. The last IC used is a 74LS04 hex inverter which provides the proper level on the strobe line to the 8255 and also to invert the INTR signal generated by the 8255 for use by the 64180. In order to better understand how the circuit accomplishes what it does, and how the timing diagram for Mode 1 input operation affects the circuit, a brief discussion about the 8255 is in order.

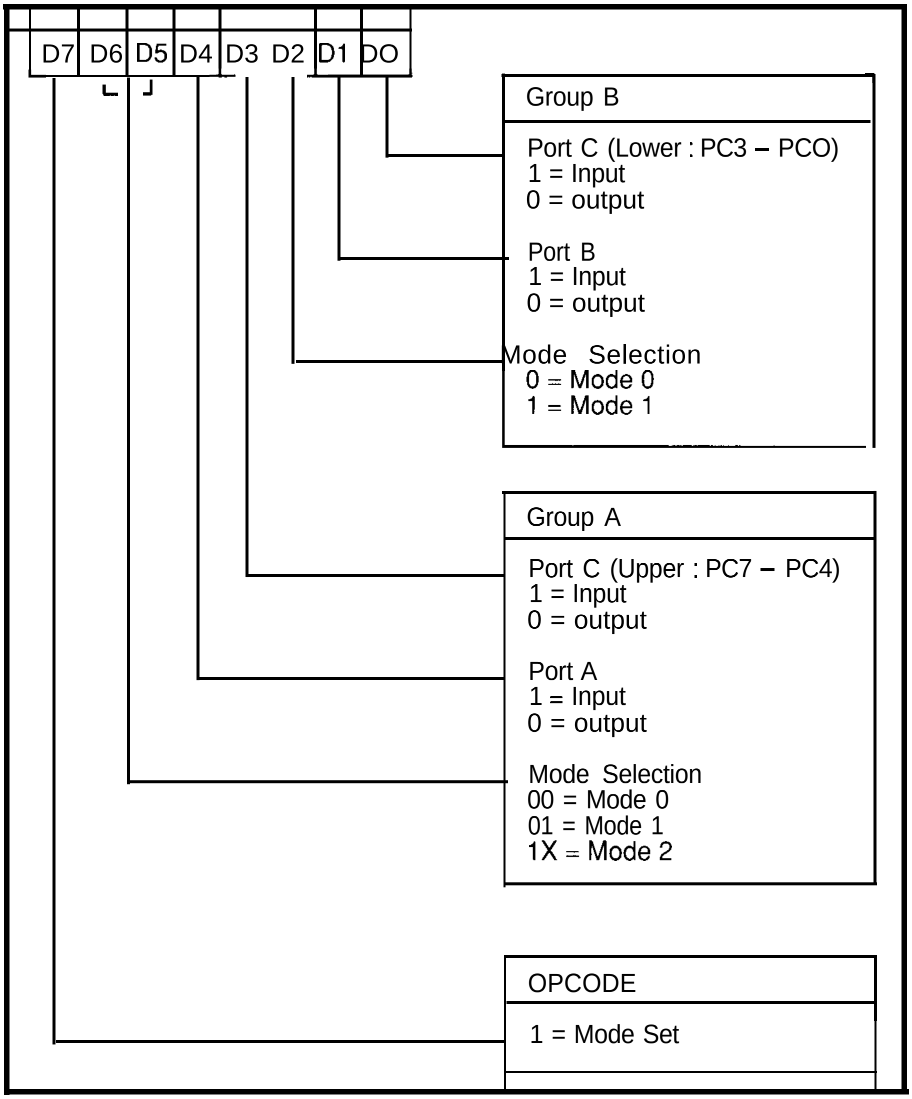


Figure **2**—*When* bit 7 *of the Control Register is set, the register is used to configure pork A, B, and C for input or output operation and also sets the overall operating mode for the chip.*

The 8255 PPI in its simplest form, connects to any given microprocessor or microcontroller via an 8-bit data bus, two address lines, a chip select, and read/write control signals. The connection to the outside world is accomplished through three ports referenced as Port A, Port B, and Port C. Three operational modes work to configure the characteristics of these ports in different ways. See Figure 1 for a quick reference on these modes.

Mode 0 is the basic I/O mode which divides the 24 I/O lines into two 8-bit bidirectional ports (ports A and B) and two 4-bit subgroups for port C, known as Group A and Group B. Mode 0 is also the simplest of the three modes available and provides latched outputs for those ports that are configured as outputs. The ports configured as inputs act as tristate buffers and are not latched.

Mode 1 provides "strobed" I/O via unidirectional ports A and B configured as inputs or outputs. Handshaking signals configured in port C provide the strobes to gate data in and out of ports A and B, which are latched for both inputs and outputs.

Mode 2 is similar to Mode 1 because it provides strobed I/O signals from port C, but differs in that only port A is used and operates as a bidirectional port.

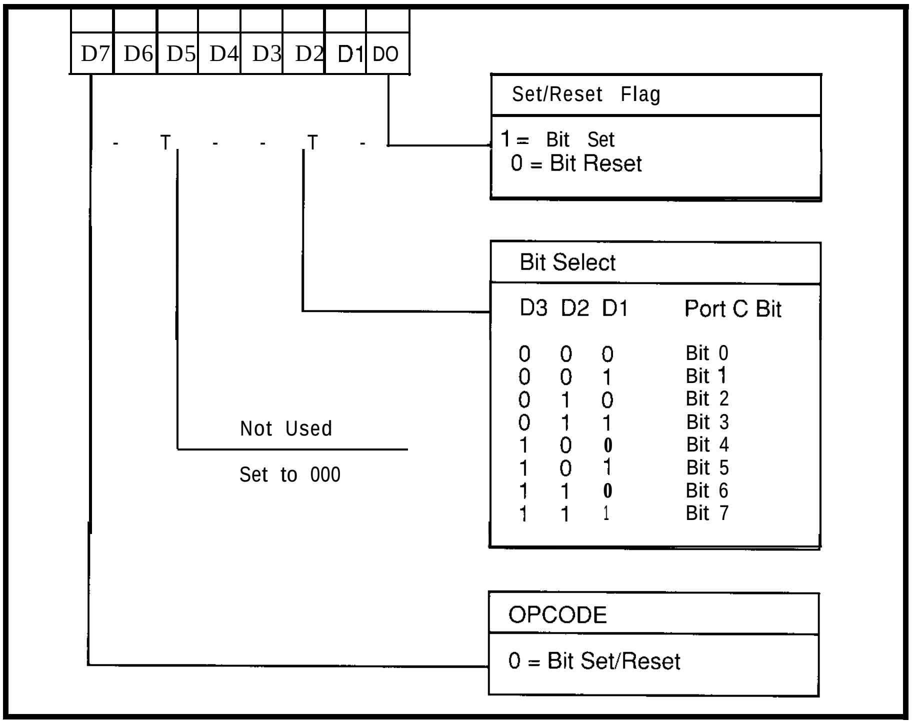
Address lines A0 and Al provide the means of accessing the data for each port and configuring the 8255's operation. The four I/O port addresses (as seen from the CPU) are defined as follows:

port n-Port A
port n+l-Port B
port n+2—Port C
port n+3—Control Register

where *n* is some address starting on a 4-byte boundry. On the board I used, *n* = 8000h. Figure 2 shows how to set the Mode Control register, while Figure 3 shows the Bit Set/Reset Control register. Both registers are accessed at port *n*+3.

A control byte of B0h configures the 8255 completely for Mode 1 input operation using port A as an 8-bit input port, port B as an 8-bit output port, and port C as control signals for port A and auxiliary I/O lines.

Figure 4 shows the Mode 1 input timing for the 8255 and Figure 5 shows the basic schematic for the circuit.



D3	D2	D1	Port C Bit
0	0	0	Bit 0
0	0	1	Bit 1
0	1	0	Bit 2
0	1	1	Bit 3
1	0	0	Bit 4
1	0	1	Bit 5
1	1	0	Bit 6
1	1	1	Bit 7

Figure 3-When *bit 7 of the Control Register is clear, the register is used to set or clear individual bits on port C without affecting the rest of the bits on the port.*

During the initialization process, PC7, which is tied to U2/13(● CLR) and U2/4 (*PR), is brought low to set the Q output at U2/9 low and the Q output at U2/5 high. This sets up the dual flip-flop to produce an alternating reset signal for the LS280 (Ul). Just prior to enabling *INTO, PC7 is set high again to allow the LS74 to function.

When an odd number of active-high signals appears on the inputs A thru H (including the input on pin 4), the Odd parity signal at pin 6 will go high. This transition clocks the 8255's ● STB signal (PC4) to indicate that there is new data to be latched. It also strobes the first flip-flop stage and causes the output of the first stage to flip to the opposite state. The 8255 responds to the ● STB signal by asserting its INTR line (PC3), which is inverted and fed back to the 64180's *INTO input.

When 'STB goes low, it also causes the 8255's IBF (Input Buffer Full) signal to go high when the data on PAO-PA7 has been latched. When IBF goes high, it clocks the second flip-flop stage, causing the input on U1/4 to flip states, restoring even parity and effectively resetting the LS280.

Finally, when the 64180 responds to the interrupt by reading port *n* (port A), the falling edge of ● RD resets the

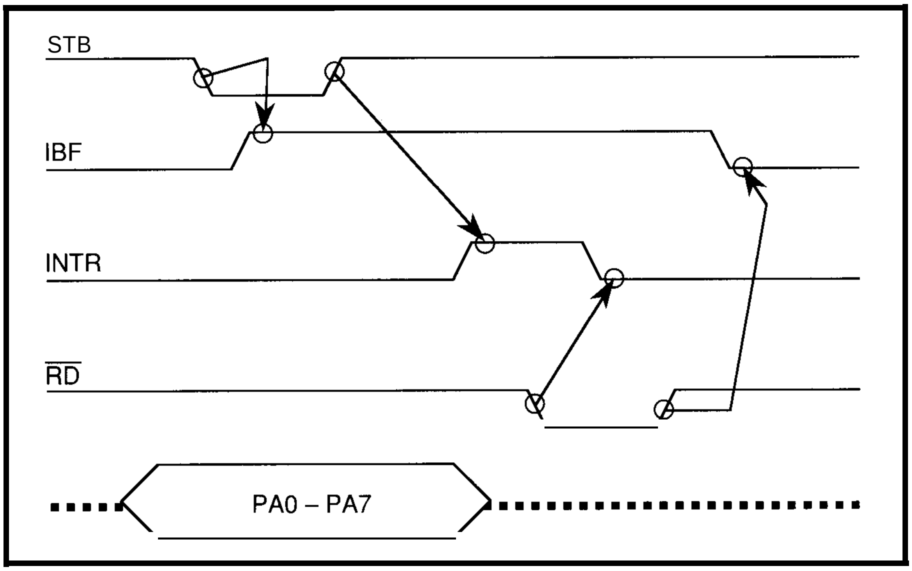


Figure 4—When the 8255 is in mode 1 and the STB* input goes active, the IBF (input buffer full) output goes active to indicate the data has been latched. When STB* goes away, the 8255 signals an interrupt by pulling INTR active. Reading the data from the chip c/ears the interrupt and IBF.

8255 INTR signal and the rising edge of 'RD resets IBF. The circuit is now ready for another input bit to change state.

1'11 demonstrate the use of the interrupt-driven parallel port by simply connecting eight push buttons to port A and eight LEDs to port B. Whenever a button is pressed, the controller lights the corresponding LED. I realize you don't need a complete controller to accomplish such a trivial task, but it serves well as an example.

THE SOFTWARE

The software to support the interrupt-driven parallel port is simple because we've done away with the polling and have purposely selected an easy example. I'll skip the basic chip initialization code because you presumably already have that in your system.

The first step is to set up the input port interrupt service routine. You could set up the routine to actually act on the incoming data, but I chose to simply read the port, save it in memory, and set a "data available" flag. The code actually jumps over the ISR to begin the initialization. The start of the initialization section programs the 8255 for Mode 1 with port A as input, port B as output, and port C auxiliary I/O pins as output. PC7, which is one of the I/O pins in port C, is set low to "preset and clear" the LS74 (U2). The internal interrupt enable for port A is set, and a zero byte is sent to port B to turn off all LEDs. Before INTO is enabled, the status of the IBF bit in the Mode 1 status word

Listing 1 -By *puffing the parallel port processing in the background, fhe foreground code can attend to more important matters.*

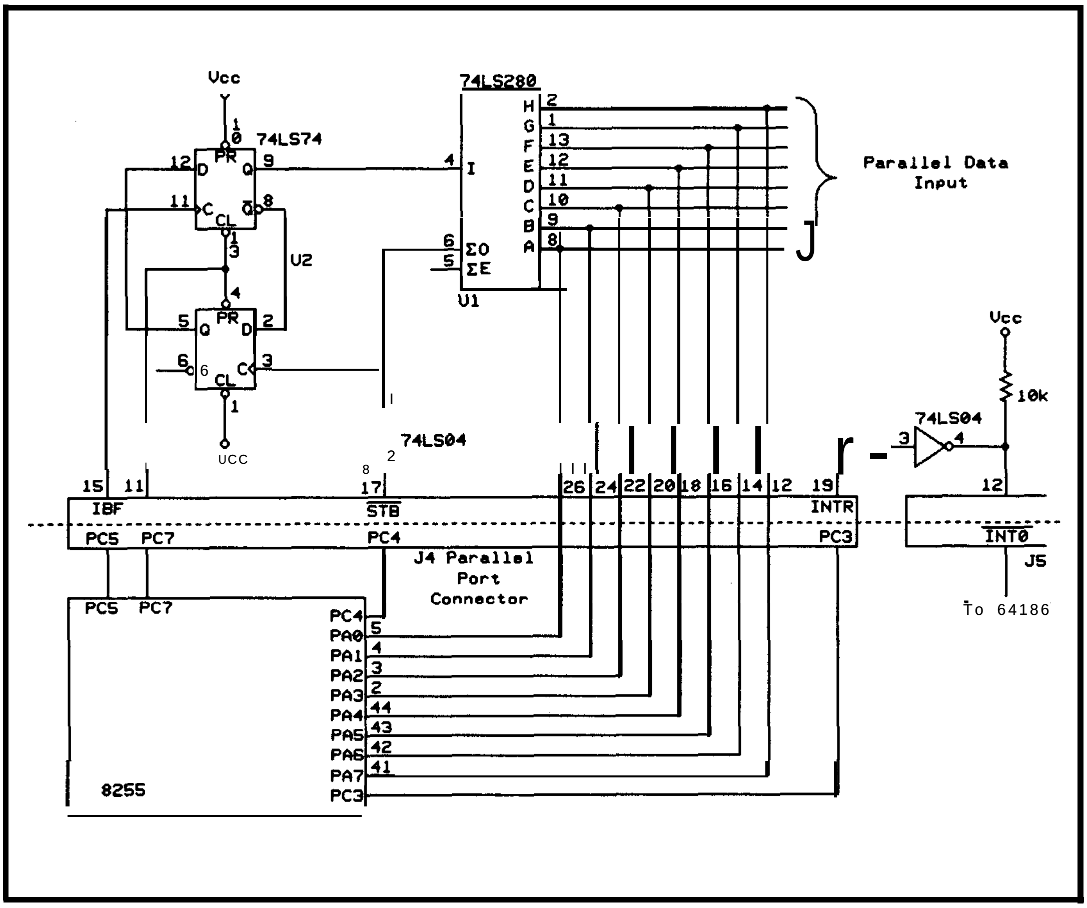
```
; Interrupt Control

ITC          equ     34h          ; INT/TRAP Control Reg.

; I/O Ports

ppi180_a     equ     8000h        ; Port A data register
ppi180_b     equ     8001h        ; Port B data register
ppi180_c     equ     8002h        ; Port C data register
ppi180_ctrl  equ     8003h        ; 8255 control register

ramstart     equ     08000h       ; Start of RAM

stack        equ     8A00h        ; Stack start location

. ***********************************************************

; Interrupt Vectors

; ***********************************************************

             org     0000h

; Power On Reset

             jp      maininit     ; Jump to Init

; INTO Maskable Interrupt 0 in Mode 1

             org        0038h

             push    af
             push    bc
             ld      bc,ppi180_a  ; Read Port A of 8255
             in      a,(c)
             ld      (portdata),a; Store port data
             ld      a,1
             ld      (datav),a    ; Signal data available
             pop     bc
             pop     af
             ei
             ret

; ***********************************************************

; Device Initialization

; ***********************************************************

             org     0100h

maininit:
             di
             ld      sp,stack     ; Set stack pointer
             im      1            ; Set Interrupt Mode 1
             xor     a
             auto    (ITC),a      ; Disable All Maskable
                                          Interrupts
; Initialize 8255 PPI

; 0B0h Port A: Mode 1 Input
         Port B: Mode 0 Output
         Port C: PC6, PC7 Output

             ld      a,OBOh       ; Initialize PPI
             ld      bc,ppi180_ctrl
```

(continued)

Listing 1-continued

```
         out      (c),a

         ld       a,00001110b  ; Reset PC7 for circuit setup
         ld       bc,ppi180_c
         out      (c),a

         ld       a,00001001b  ; Enable Intr A
         ld       bc,ppi180_ctrl
         out      (c),a

         xor      a            ; Turn off all LEDs
         ld       bc,ppi180_b
         out      (c),a

         ld       (portdata),a; Initialize variables
         ld       (datav),a

; Signal End Of Initialization

         ei                    ; Enable Interrupts
         ld       bc,ppi180_c  ; Read Port C status of 8255
         in       a,(c)
         and      00100000b    ; Isolate IBF, intr already?
         jr       z,ibfok      ; No, IBF in ready state
         ld       bc,ppi180_a  ; Read Port A to reset IBF
         in       a,(c)
ibfok:
         ld       a,00001111b  ; Release circuit setup
         ld       bc,ppi180_ctrl
         out      (c),a
         ld       a,01h        ; Enable Interrupt 0 to work
         out0     (ITC),a
; ***********************************************************

; Main Loop

; ***********************************************************
topml:
         ld       a,(datav)    ; Test "data available" flag
         cp       1
         call     z,procinp    ; Process input port data

                               ; Main Body of Program
         jr       topml

; ***********************************************************
; procinp gets the 8-bit data held at portdata and outputs it
;          to port B which is tied to the display LEDs. The
;          data available flag is reset.
:procinp:
         ld       a,00h        ; Disable Interrupt 0
         out0     (ITC),a
         ld       a,(portdata); Get port data
         ld       bc,ppi180_b  ; Output data to port B
         out      (c),a
         xor      a
         ld       (datav),a    ; Reset "New Data" available
         ld       a,01h        ; Enable Interrupt 0
         out0     (ITC),a
         ret                   ; End Sensor Input Processing
; ***********************************************************
         org      ramstart

portdata ds       1            ; Port A input data
datav    ds       1            ; Data available flag

         end
```

Figure 5—*When* a *bit of input* data *changes, the 74LS280 parity checker signals the change to the 8255. The 8255 latches the data and generates an interrupt to the processor. When the processor reads the data from the 8255, the interrupt is cleared and the circuit is reset to be ready for the next bit change.*



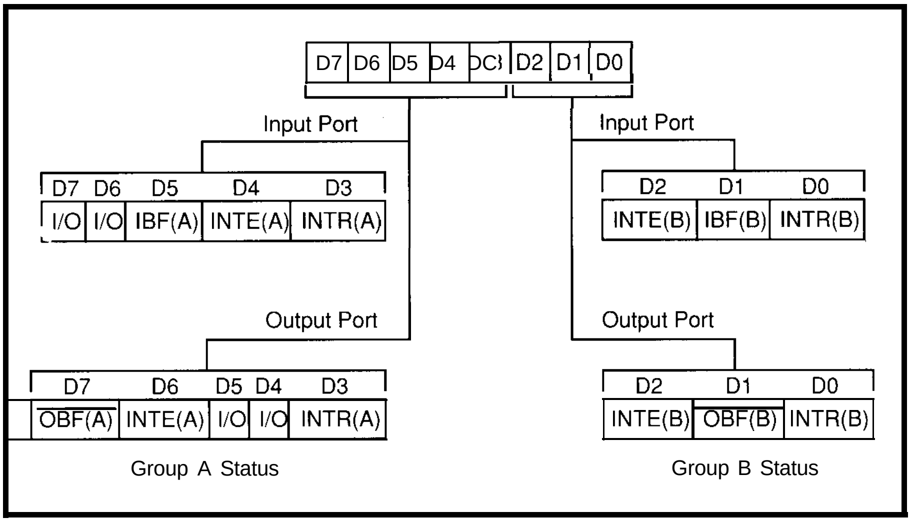


Figure 6—*In modes 1 and 2, port C provides all hinds of status information and control bits for ports A and B.*

(port C read) is checked to make sure an interrupt is not pending. If so, the port is read to clear it before enabling INTO and the LS74 flip-flop. If all is clear, U2 and INTO are enabled.

The main loop tests a "data available" flag which is set in the INTO routine. If set, the input processing routine is called, otherwise the main loop is free to do other work.

GOING FURTHER

The 8255 is such a versatile chip, it would take a book to describe all the different ways to configure and use it. The book I've used for the last 10 years when dealing with the 8255 is "Microcomputer Interfacing with the 8255 PPI Chip." I don't have room here to describe the many different uses for all three modes, let alone how many different ways Mode 1 by itself could be used.

The 8255 has been around for a long time, and for good reason. It continues to meet the need of providing a flexible design that is able to fullfill the requirements of many different applications without having to scrap and redesign. ◙

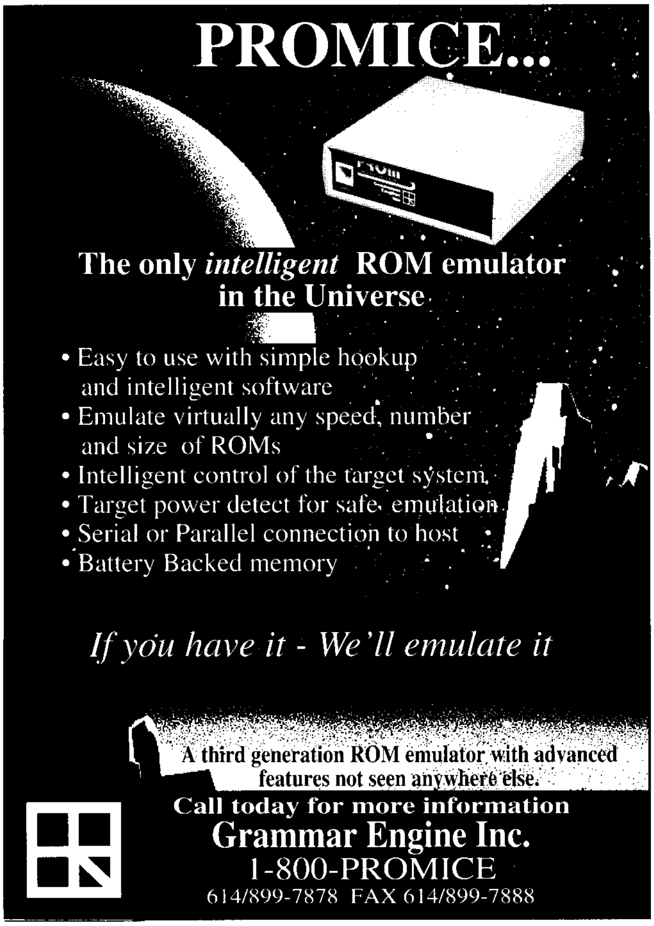
James ***Grundell has been working on computer hardware and software for*** *14* ***years and has experience in building automation and avionics.*** *He* ***may be reached on CompuServe as*** *76167,45.*

REFERENCE

"Microcomputer Interfacing with the 8255 PPI Chip," Paul F. Goldsbrough, Howard W. Sams & Co. Inc., 1980.

I R S

410 Very Useful
411 Moderately Useful
412 Not Useful



DEPARTMENTS

Chickens and Eggs: The First Embedded '386SX Programs

The quest for embedded code running on a '386SX motherboard continues as Ed explores basic development tools, startup code, console I/O, and the first real embedded C program.

FIRMWARE FURNACE

Ed Nisley

Now that we have a way to boot programs into our '386SX system, what should the First Program be? A multitasking, multiuser, multimedia protected-mode operating system featuring 32-bit flat memory model programming with no artificial segment limits?

Hardly!

The whole point of this series is to make sure you know how to write good firmware. While I suppose you can simply ignore all of the low-level details, I've found that sort of knowledge is indispensible in ferreting out bugs, taking maximum advantage of the hardware, and generally doing a good job.

The First Program should be a debugger. The Second Program should be a simple C program. After that, we can look at the hardware we'll need to do some useful work. After that.. .well, all things in due time.

Despite the killer GUI debuggers now in vogue for PC program development, a traditional command-line debugger has a lot to recommend it for simple projects. Those "primitive" debuggers are relatively small, you can drive them from a serial port, and they do what you need to get things started. If you have ever used the DOS DEBUG program (and who hasn't?) you know what I mean.

But, because the '386SX board doesn't have DOS, we can't use good old DEBUG. Rather than write my own debugger, I'll take advantage of Dunfield Development Systems' 8086 Micro-C Developer's Kit. It includes the Micro-C compiler, an assembler, two debuggers, and a host of other utilities useful for embedded '86 program development.

There are, of course, higher-powered ANSI C/C++ compilers that can generate code for embedded '386 CPUs and ways to get GUI debuggers running through serial ports. Dunfield's programs are straightforward enough that we can see the fundamentals quite clearly. In upcoming columns, I'll get more complex, but, for now, let's get the basics running.

THE BARE ESSENTIALS

The diskette boot loader I described last month is a simple and cheap way to load a program into the '386SX system. One of our first projects will be a firmware development board with some EPROM to make the diskette optional, but the code from this column will help check out that board in style.

Recall that the '386SX system includes an integrated I/O board with a pair of serial ports. Because we are not using the PC's keyboard or display, those two ports are an ideal way to communicate with the firmware. It seems a shame to let a resource like that go to waste, so I'll use COM1 for the firmware and COM2 for the debugger needed to get the code running.

Dunfield's HDM86 (which stands for Hardware Debug Monitor) is one of the slickest chunks of code I've seen in a while. It will run on any 80x86 CPU and uses no external RAM: all of the data stays in the CPU's registers, including the return address for one level of subroutine call!

Although you won't need HDM86 for our projects, it comes in handy to verify hardware startup functions such as DRAM refresh and memory mapping. Of course, HDM86 must be in EPROM and you can't download any code (where would it go?). Performing the actual I/O operations "by hand" can often reveal circuit problems.

HDM86 includes commands to put the code in a tight read/write loop hammering on an I/O or memory address. This loop makes it easy for you to scope out problems because the hardware strobes are convenient sync points. Of course, you have to reset the system to regain control, but that's why HDM86 lives in EPROM.

Incidentally, CPU and system board hardware setup is one of the (many] problems I've avoided by using a '386SX system board. As far as I'm concerned, the BIOS sets up everything and passes control to me through the disk boot loader. While I can change anything necessary, the essential startup details are a done deed.

Remember, the rules are changing. You give up detailed hardware diagrams to gain absolute PC compatibility and a rock-bottom per-unit price. All things in PC-Land are negotiable. You can get schematics if you're using system boards in production quantities and you can write your own BIOS if you need absolute control. Relax: we aren't lowering the net very far!

MON86, the larger of Dunfield's two 8086 debuggers, occupies a little over 5K (gasp!) and includes a disassembler, hex file loader, eight code breakpoints, and the usual DEBUG-style commands. It arrives set up for the PC's keyboard and video display, but is easy enough to adapt to any standard PC serial port. Dunfield includes a detailed set of comments to help you through the process, which involves changing a few constants and reassembling the code.

The diskette boot loader handles standard COM programs starting at offset 0100, but MON86 assumes its programs begin at 0000. The only change needed is to set MON86's initial User PC (Intel fans know it as the Instruction Pointer: IP) to 0100 instead of 0000. MON86 may include this feature by now, with comments to explain it.

MON86 will not display '386-specific instructions, but this is not a

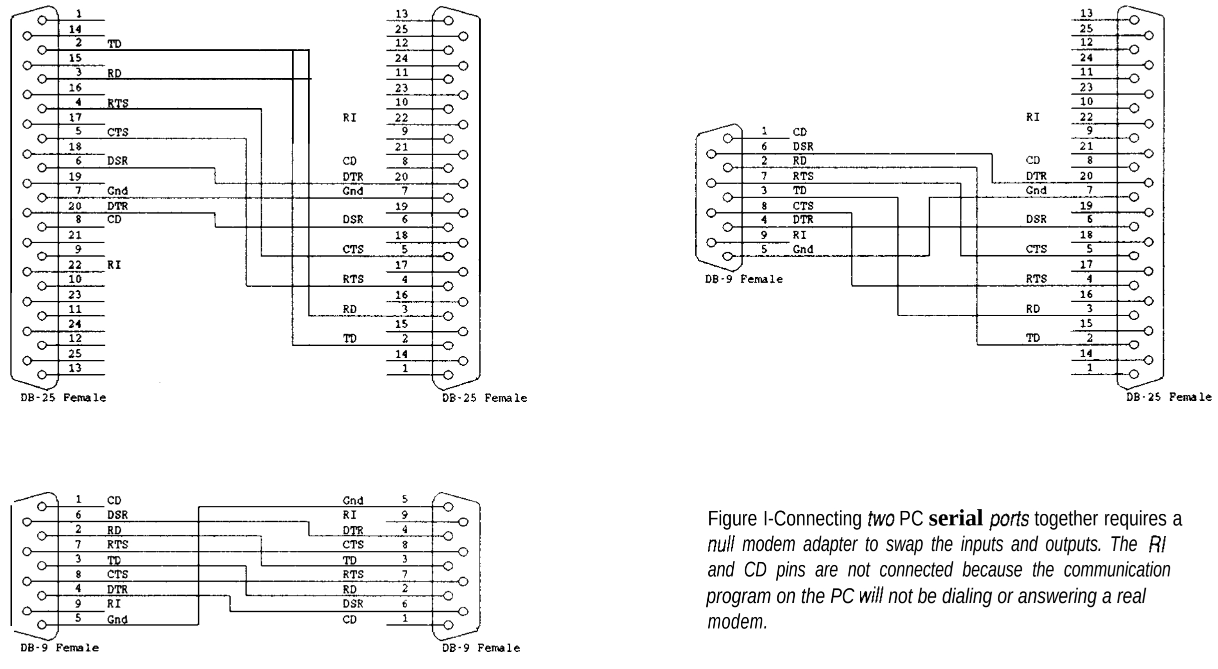


Figure I-Connecting *two* PC **serial** *ports* together requires a *null modem adapter to swap the inputs and outputs. The RI and CD pins are not connected because the communication program on the PC will not be dialing or answering a real modem.*

Listing 1—*Use MON86 to load ROTATE.HEX through the serial port. The LED and switch hardware described last month will let you see the results on the parallel printer port. The code shown here should be assembled with Dunfield's ASM86 assembler; the syntax is slightly different than Microsoft's MASM or Borland's TASM, but no more unusual than some other assemblers*

```
         ORG       $100

         MOV       DX,#$0378     Select parallel port
         MOV       AL,#$80       Set up the rotating bi
*
ReShow   ROL       AL.1          Rotate it
         OUT       DX,AL         And display it
*
         PUSH      DX            Save our values
         PUSH      AX

         MOV       CX,#1         Use BIOS delay function
         MOV       DX,#0            to get 64K microseconds
         MOV       AH,#$86
         INT       $15

         POP       AX            Recover our values
         POP       DX

         JMP       <ReShow
```

serious impediment during the first few projects because plain old 8086 instructions have the functions we need. When '386 instructions will give us a major performance boost, I'll shift to an assembler and debugger that support them directly.

LOAD AND GO

You could burn the modified MON86 into EPROM, but I put it on a diskette and booted it into the target system's RAM using the loader from last month. Remember that the whole point of this exercise is to build the tools to build the board to hold that EPROM!

You will need a serial port and communications program on your main PC to talk with MON86 on the target '386SX. Figure 1 shows three possibilities for the "null modem" connections needed to connect two PC serial ports.

When MON86 starts up on the target system, you should see a display similar to what's shown below:

```
MON86 Version 1.1
Copyright 1991,1992 Dave Dunfield
All rights reserved.
*
```

If not, recheck the source in MON86.MAC to verify that all the options are correct.

Although I shouldn't have to mention this, make sure you plug the serial cable into the right ports on both PCs. I speak as an expert, having gotten both ends wrong on the first crawl under the desk.

Once MON86 boots up, use the Output command to write a few values to the parallel printer port at address 0378. The LED and switch circuit from last month's column will come in handy to watch the bits toggle. This simple step verifies that MON86 is loaded, the serial ports and your comm program work, and the parallel port is at the expected address.

The next step is to load a hex program with the Load command. ROTATE.ASM, shown in Listing 1, is a simple loop that rotates a single bit across the parallel port. It uses a BIOS function to make the output eyeball-visible so you won't need a scope to check this code out.

Dunfield's assemblers can produce either Intel or Motorola hex files and MON86 will accept either flavor without any special attention.

MON86 does not echo the characters it receives while transferring the hex data, so you may have to configure your comm program to not wait for the end of each line. If the comm program seems to hang after sending the first line, the two programs are probably locked in this deadly embrace. Cancel the transfer, reset the comm program's options, and try again.

Dunfield's HEXFMT utility will transform ROTATE.HEX into ROTATE.BIN, which the diskette boot loader will pull directly into RAM. The main advantage of changing MON86's default User PC is that you can debug programs and run them without change.

The last piece of the puzzle is the C compiler. As it turns out, the effort is almost anticlimactic because most of the work is already done.

MICRO-C STARTUP

Back in issues 23 and 24, I described the C startup code that runs right after an 8051 CPU comes out of a hardware reset, before main() gets

Listing 2—*The Micro-C startup code for the Tiny memory model simply loads the segment registers and calls the main() function. Micro-C does not preload 'uninitialized" variables with zeros, but anyone using a variable without setting if first deserves the results!*

```
            ORG       $100
*
Restart     MOV       AX,CS         Get CODE segment
            MOV       DS,AX         Set DS
            MOV       SS,AX         Set SS
            MOV       SP,#$FFFE     Initial stack pointer
            MOV       <?heap,#0     Zero heap
            CALL      main          Execute main program
*** Restart if we fall out the bottom; should not happen!
            JMP       <Restart
* Exit function... Return to MSDOS
exit        XOR       AH,AH         Function O-exit
            INT       $21           Exit
```

Listing 3—*Micro-C's default console I/O routines use the BIOS video and keyboard functions. This code substitutes the corresponding BIOS serial I/O functions, which, while nof adequate for heavy-duty use, are good enough to get our first few programs running.*

```
*
*Set up the serial port
*  serinit(bitrate,COMport)
* Assumes 8N1 format
*
serinit MOV      BP,SP        Get pointer 'co params on stack
        MOV      BX,DS        set up ES for SCAS
        MOV      ES,BX
        MOV      AX,4[BP]     fetch bit rate (stacked first)
        MOV      CX,#8        Set table length
        MOV      DI ,#?bittab Set table base
        REPNE
        SCASW                 Search for rate
        MOV      AL,CL
        NEG      AL           Convert count to table index
        ADD      AL.#7         (if not found, use last entry)
        MOV      CL,#5        Move bits 2:0 to 7:5
        ROL      AL,CL
        AND      AL,#$E0      clean out. low bits
        OR       AL,#$03      Set no parity, 1 stop, 8 data
        MOV      AH,#0        BIOS function number
        MOV      DX,2[BP]     Get COM number (stacked last)
        DEC      DX           Make COM1 = 0
        MOV      ?COMport,DX  save for later
        INT      $14          Do the BIOS initialization

* set up modem control outputs to ensure good communications
*
        MOV      BX,?COMport  get port index
        ADD      BX,BX           convert to word index
        MOV      AX,#$0040    aim ES at BIOS data area
        MOV      ES,AX
        MOV      DX,ES:[BX]   fetch port address
        ADD      DX,#$0004    aim at modem controls
        IN       AL,DX        fetch current bits
        OR       AL,#$03      force RTS & DTR high (inverted)
        OUT      DX,AL
        RET


* Table of BIOS-acceptable bit. rates
* The entry index is the BIOS bit rate code value
*
?bittab DW       110          0
        DW       150          1
        DW       300          2
        DW                    3
        DW       1200         4
        DW       2400         5
        DW       4800         6
        DW       9600         7
*
* Write a string to the serial port
*
putstr  MOV        BX,SP      Address stack
        MOV        SI,2[BX]   Get string

?putstr MOV        AL, [SI]   Get char
        INC        SI         Skip to next
        AND        AL,AL      End of string?
        JZ         ?1         Yes, exit
        CALL       ?putch     Write it
        JMP        <?putstr   Do it all
*
* Write char with NEWLINE<>CR translation
*
putch   MOV        BX,SP      Address stack
        MOV        AX,2[BX]   Get character

?putch  CALL       ?putchr    Write value
        CMP        AL,#$0A    Newline?
        JNZ        ?1         No, skip it
        MOV        AL,#$0D    Add carriage return
        JMP        <?putchr   Write it out
*
* Write char with no translations
*
putchr  MOV        BX,SP      Address stack
```

(continued)

control. Regardless of whether you use an 8051 or one of the ‘86 family, you must understand the how, what, and why of that startup code.

The “bare metal” startup begins when an ‘86 CPU emerges from hardware reset by fetching the first instruction. Each of the 8086, 80286, 80386, and 80486 CPUs start with slightly different CS:IP register values, but the system board generally maps the resulting physical address to FFFFO (i.e., F000:FFF0).

As I mentioned earlier, the PC’s BIOS handles all the initialization required to get from that first instruction fetch to a system that can boot an operating system-or your embedded program. If you are writing ‘86 embedded code without the luxury of a BIOS, you must know the details of the hardware so you can provide all the right startup functions.

Several vendors offer “BIOS kits” to reduce the tedium of rolling your own BIOS. Even if you don’t need all the standard functions, starting with known-good code can trim months off your schedule. Check the ads for sources; this is a small but growing niche market.

For our present purpose, however, the BIOS sets up the hardware, reads our boot loader from the diskette, sets up the segment registers, and passes control to it. Our loader reads in the first DOS file, sets up the segment registers, and jumps to whatever is at offset 0100. That’s where the C startup code must begin.

Listing 2 shows the top part of 8086RLPT.ASM, Micro-C’sTiny memory model startup code. It sets up the segment registers (again!) and starts the C program with an LCALL to the main() function. If main() returns, the default startup code invokes the INT $21 “terminate program” DOS function.

I replaced the termination code with a simple JMP back to the top, but even that isn’t strictly necessary. As you will see later, our embedded C programs never end, so that JMP should never be executed. Nevertheless, I feel better with a valid instruction instead of a guaranteed trip to the weeds; call me compulsive.

Listing *3-continued*

```
        MOV       AX,2[BX]  Get character
*
* Write through IBM/PC BIOS call
*
?putchr MOV       DX,?COMport use our port
        MOV       AH,#$01   Write char to port
        INT       $14                 Call BIOS
?1      RET
*
* Hang until a character arrives
* Return value is raw character in At

waitchr MOV       AH,#$03             Status request
        MOV       DX,?COMport         use our port
        INT       $14                 Ask BIOS
        TEST      AH,#$01             Check data ready
        JZ        waitchr             Char is ready
        MOV       AH,#$02             Get character
        MOV       DX,?COMport         use our port
        INT       $14                 Ask BIOS
        XOR       AH,AH               clear high byte
        RET

* Test for character (No translation)
*
chkchr  MOV       AH,#$03             Status request
        MOV       DX,?COMport         use our port
        INT       $14                 Ask BIOS
        TEST      AH,#$01             Check data ready
        MOV       AX,-1               assume no char
        JZ        ?ccr                correct, bail out
        MOV       AH,#$02             Get char
        MOV       DX,?COMport         use our port
        INT       $14                 Ask BIOS
        XOR       AH,AH               discard flags
?ccr    RET

* Receive char with no translations
*
getchr  CALL      waitchr             Wait for char and get it
        RET

* Test for character (with CR<>NEWLINE translation)
*
chkch   MOV       AH,#$03             Status request
        MOV       DX,?COMport         use our oort
        INT       $14                 Ask BIOS'
        TEST      AH,#$01             Check data ready
        MOV       AX,#0               assume nothing
        JZ        ?cch                correct
        MOV       AH,#$02             Get character
        MOV       DX,?COMport         use our port
        INT       $14                 Ask BIOS
        XOR       AH,AH               discard flags
?cch    RET
*
* Receive char with CR<>NEWLINE translation

getch   CALL      waitchr             Wait for char and get it
        XOR       AH,AH               Zero HIGH
        CMP       AL,#$0D             Carriage return
        JNZ       ?1                  No, its OK
        MOV       AL,#$0A             Convert to NEWLINE
        RET
*
* Receive a string: getstr(buffer, size)
*
getstr  MOV       BX,SP               Address stack
        MOV       SI,4[BX]            Get buffer address
        XOR       cx,cx               Zero CX

?2      CALL      getch               Get character
        CMP       AL,#$08             Backspace?
        JZ        ?3                  Yes, handle it
        CMP       AL,#$7F             Delete?
        JZ        ?3                  Yes, handle it
        CMP       AL,#$0A             End of line?
        JZ        ?4                  Yes, handle it
        CMP       CX,2[BX]            Are we at end?
        JAE       ?2                  Yes, don't accept
```

(continued)

Reloading the segment registers three or four times in quick succession is esthetically displeasing, but of no real consequence because the values are the same. The way I've set things up, each step in the chain is independent of the others, so the register loads are needed Just In Case the previous program isn't under my control.

Because the diskette boot loader is limited to COM files, all the code and data must reside in one 64K-byte segment. This implies all the segment registers have the same value, which is precisely the definition of the Tiny memory model. Micro-C can also produce Small model code, but we can get along quite happily with a single segment for a while.

CONSOLE I/O

The Micro-C runtime library includes the usual console I/O functions that connect getch() and putch() with the outside world. Despite the fact that they're in a file called SERIO.ASM, getch() comes from the PC keyboard and putch() *goes* to the video adapter. This mapping makes a lot of sense for most applications that use PC hardware, but we don't yet have the luxury of standard PC I/O.

Fortunately, SERI0.ASM uses the BIOS keyboard and video functions, so it's easy to substitute the corresponding serial I/O functions. While the BIOS code is not adequate for "real life" serial I/O, it will suffice for our immediate needs. Higher performance is a simple matter of software that we can work on when we need it.

Listing 3 shows the "Firmware Furnace" console I/O routines in SERIOFF.ASM. Use SLIB, Micro-C's library utility, to replace SERIO.ASM inTINY.LIB:

```
slib i=tiny.lib a=serioff.asm
        r=serio.asm
```

Your code must call serinit() to specify the bit rate and COM port number. The data rate is limited to 9600 bps because serinit() uses the older setup function included in every BIOS ever made. If you know your BIOS supports function INT 14, AH=

04, you can run the bit rate up to **19200** bps.

Although serinit() can set up any one of the four standard serial ports, the integrated I/O card has only two ports and MON86 uses COM2. You may use the same port for both your C program and the debugger, but the output can be confusing at times.

Finally we are ready for a C program!

Listing **3**—*continued*

```
        MOV      [SI],Al           Write it
        INC      SI                Advance pointer
        INC      CX                Advance length
        CALL     ?putch            Write it out
        JMP      <?2               And proceed

* Delete character from buffer

?3      AND      CX,CX             At beginning?
        JZ       ?2                Yes, ignore

        MOV      AL,#$08           Backup
        CALL     ?putch            output

        MOV      AL,#' '           Space over
        CALL     ?putch            output

        MOV      AL,#$08           Backup
        CALL     ?putch            output

        DEC      SI                Backup pointer
        DEC      CX                Reduce length
        JMP      <?2               And proceed

* Newline, terminate entry
*
?4      CALL     ?putch            Echo it
        MOV      >[SI],#0          Zero terminate
        MOV      AX,CX             Return length
        RET
```

SAY "HELLO!"

Back before C compilers (excuse me, "application program development environments") occupied 50 megabytes of disk space, the first C program had one essential line:

```
printf("Hello, world!\n");
```

Fortunately for us, Micro-C on our embedded '386SX still retains that charming simplicity. HELLO.C, shown in Listing 4, uses printf() to send out the obligatory message through the serial I/O routines described above.

My HELLO.C cannot "fall out the bottom" because there is no operating system to regain control. Essentially all embedded programs run continuously from power-on to shutdown. Of course, high-octane embedded systems require real operating systems with dynamically loaded programs; we're not there yet!.

HELLO.C uses a while (1){...} loop to hold the "do-forever" code. In this case, I increment a counter, display the value, and call a BIOS function to waste about half a second. The delay is essential because there is no point in sending continuous 9600-bps data to the screen: you just can't read that fast!

Micro-C includes a "Command Coordinator" that runs the preprocessors, compiler, assembler, linker, and so forth in the right order with the right files. I use this command line:

```
cc86 hello -cimop m=t
```

The result is HELLO.HEX in Intel hex format. Boot MON86 from the floppy in the '386SX system and use the Load command. Send HELLO.HEX with your comm program's ASCII file transfer function, then do a G 0100 to start HELLO.

Because MON86 uses COM2 and **HELLO** uses COM1, you need two serial ports on your PC to monitor the action. This is one of the big advantages of a multitasking operating system like OS/2: simply start two comm program sessions and watch their windows update simultaneously.

You can also use a PC with a single port and one comm program, but you'll have to switch the cable from COM2 to COM1 after starting HELLO. You'll miss the "Hello" line, but the counter values should come through correctly.

You can use MON86 to step through your C programs at the assembler level, set breakpoints, and dump memory. After you're sure the code is solid, use HEXFMT to convert the **HEX** file to a COM file just as you did for ROTATE.HEX:

```
hexfmt hello.hex -b
      w=hello.bin
```

Copy that file to a diskette with the boot loader from last month, pop it in the '386SX system's drive, and hit the reset button. You should see the same output on COM1 as you did when running it under MON86.

At this point you've got everything you need to develop embedded programs on a decent system. The diskette boot loader gets a program into the '386SX. Micro-C has the essentials in a well-documented, comprehensible package, and the diskette boot loader eliminates the need for special hardware.

Ball's in your court!

RELEASE NOTES

I'd planned to investigate I/O performance this month, but decided to spend more time on the startup code so you have a better idea of how the code gets going in the first place. Next month I'll make bits, bytes, and words jump through hoops.

You've heard about OS/2 here before, but this column highlights why I like it so much. While writing this text with Describe (a GUI word processor], I have a DOS session (with 722K of free RAM!) for the Micro-C programs, two copies of REXXTERM for the debugger and C program serial I/O, KEDIT to edit the source code, and LISTPM to browse the compiler output, all active at once.

Listing 4—The canonical First C Program prints "Hello, world!" and returns to the operating system. Our '386SX system doesn't have an operating system, so, after this version of HELLO.C displays the obligatory message, it begins an endless loop showing the contents of a simple counter. A BIOS function provides a half-second delay between displays.

```
#include <8086io.h>
#include "e:\mc86\custom\8086base.h"
#define LOOPDELAY 8            /* units of 64K microseconds */

unsigned int Counter:

main0 {
    serinit(9600,1);           /* set up serial port */
    putstr("Hello, world!\n");
    Counter = 0:

    while (TRUE) {
        printf("Counter: %5u\r",++Counter);
        asm {
            MOV CX,#LOOPDELAY
            MOV DX,#0
            MOV AH,#$86
            INT $15
```

My time-and-phone manager runs in a background WIN-OS2 session, another copy of REXXTERM downloads messages from the Circuit Cellar BBS, and a few other programs chug along. When I need to create a schematic, I just start up Orcad in a full-screen DOS session.

Sure, you can keep running programs one at a time, but why bother? PC systems are as cheap as they'll ever be, so it's time to get up to speed. Heck, even Windows is better than DOS for this kind of thing!

The BBS files include all the demo source code, HEX, and BIN files, as well as the serial port code. There are a few useful batch files to set up Micro-C's environment variables, too.

You'll need Dunfield's 8086 Micro-C Developer's Kit to compile and assemble the source code. The debuggers described above are a part of the package, as well as numerous utility programs that I haven't mentioned. Contact DDS at (613) 256-5820 (fax [613] 256-5821 during the day) for current price and delivery information.

Of course, you don't have to use Micro-C. The disadvantage of the Borland, Microsoft, and similar high-end PC C compilers is that they are intended for use with DOS; adapting them to a DOS-less embedded system is possible, but it's not a trivial exercise. Indeed, a whole industry has grown up around this subject.

If you've already got an '86 C compiler, take a look at the startup code and see what you can do. You'll need to verify that there are no DOS calls in the generated code, which will probably rule out the normal I/O routines. Give it a go and check in on the BBS to compare notes!

Next month: 'scope shots of pretty pulses.

Ed Nisley is a Registered Professional Engineer and a member of the Computer Applications Journal's engineerig staff. He specializes in finding innovative solutions to demanding technical problems.

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

413 Very Useful
414 Moderately Useful
415 Not Useful

Seven-Segment LEDs Live On

Although LCD displays have taken over where LED displays were once used, LEDs can't be beat for their readability from a distance and in low light. Maxim now has a chip that makes driving them simple.

FROM THE BENCH

Jeff Bachiochi

“This information will be held in the strictest of confidence,” we're told, “Your providing it is just a formality.” Forms in duplicate, forms in triplicate, please press firmly to penetrate all copies.

We are all aware that the system under which we live is eroding our privacy. I try not to think about it. When I was in my teens I actually contemplated trying to exist (or not exist) without a Social Security Number. At the time, it appeared to be the only leash the government had on us. Without it, you were a nobody, and untraceable. However, once you crossed that line and applied for a SSN, it was over. Of course I can see now this was a rather paranoid view.

Today we have more numbers than we know what to do with. Take away our numbers and we lose our place in line, have no ZIP code, and can't be reached by phone or fax. We can't tell time, count, or measure. The world is united by a common language of numbers, even if we speak in different tongues. But beware, the gift of numbers is a double-edged sword. It makes our lives more comfortable, but at the same time our appetite for credit allows “them” to track our movements.

You can often pass information without the use of words. If the communicating parties have an understanding of what is being passed, numbers alone can accomplish the task. Just eyeball the cash register tallying totals or the local bank's sign giving the present time and temperature. Because displaying numbers is often less expensive than words, especially if you use seven-segment displays, using them in areas where you need to cut costs is helpful.

SEVEN-SEGMENT DISPLAY, THE ONE CHIP WAY

Perceiving danger from something you can't see is difficult. That's how I often get into trouble. You know that little saying about ASS_U_ME. At first glance, the MAX7219 from Maxim looks like a slick solution to minimizing seven-segment display hardware. And it is; just beware of the software support necessary to use the display. For some applications this approach is not the best for adding a few digits. However, the ability to append digits easily (in groups of eight] might serve you well. Also, the interface is simple. I think this circuit is one of those that calls out “build me.” Let me describe using the ‘72 19 in a simple numerical digit display project (see Photo 1).

Photo I-The *MAX7219 makes the once-cumbersome job of driving a row of seven-segment LED displays a piece of cake.*

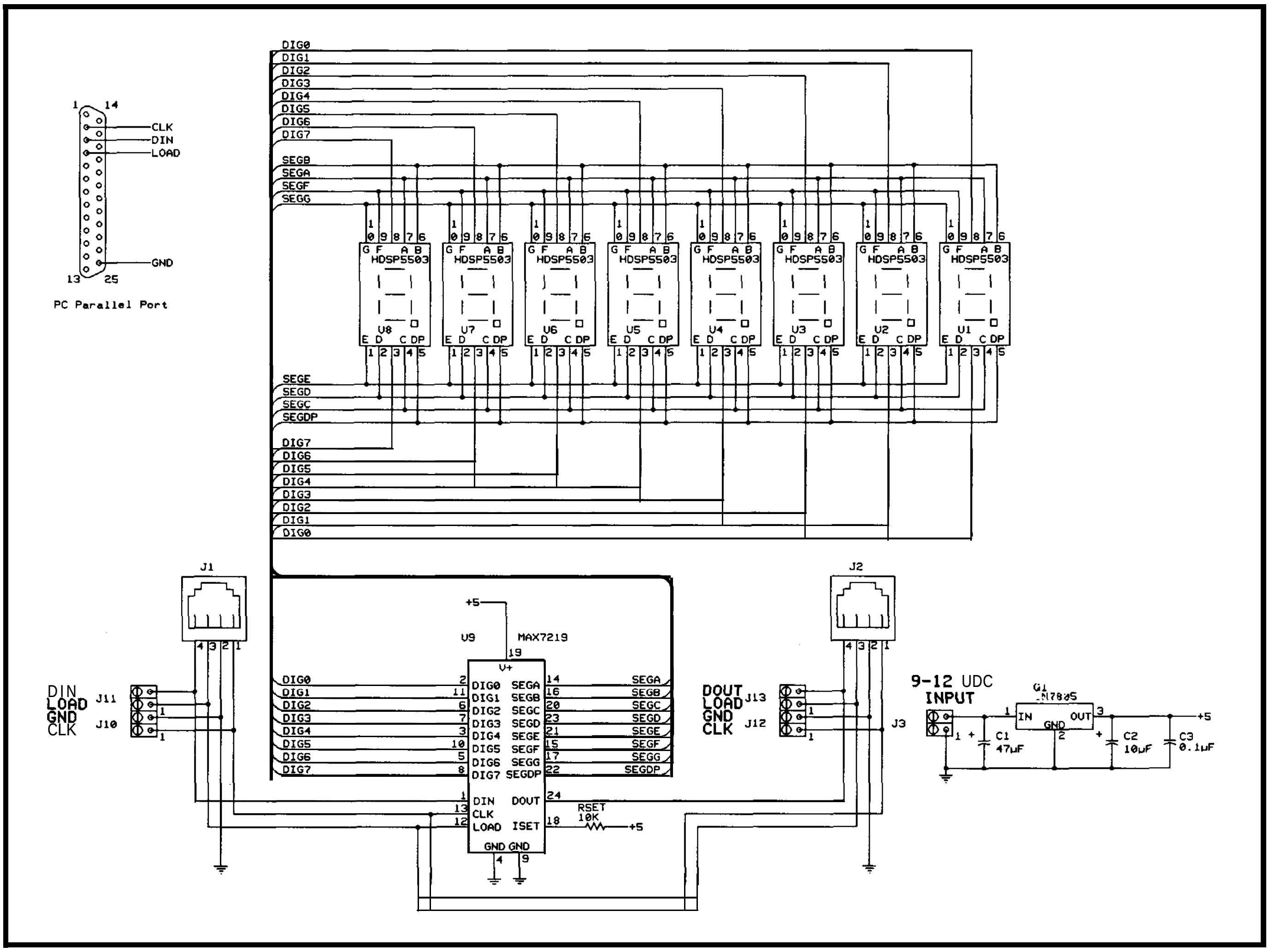


Figure I--The MAX7219 *simplifies driving seven-segment LED displays and allows you to daisy-chain multiple chips to drive very long displays*

The simple circuit in Figure 1 will permit you to control the display from your PC's parallel port or any three output pins on your favorite micro. The circuit lets you experiment with all the MAX7219's functions.

SUPERCHIP

Able to leap 64 LEDs in a single 24-pin skinny DIP, more powerful than a handful of integrated circuits, capable of displaying the course of mighty data, able to bend steel in his bare hands.. .Oops, I'm a bit choked up after reading about Superman's death. Sorry, let me get back on track.

As you can see from Figure 2, the '7219 is made up of a 16-bit shift register, a 64-byte dual-port RAM, five control registers, a PWM, a scan multiplexer, digit and segment drivers, and a segment current reference. Although the data is 16 bits in length, only the first 12 bits are used; the lower eight are for real data and the upper four for addressing. The four address bits are decoded as either one of the eight digits or one of the five control registers. Two of the remaining three decodes are not used, while the last is used as an NOP function.

The eight real data bits are saved in RAM (digits register] if the upper address bits define a digit, saved in control registers if so directed, or dumped if the address is a NOP.

FUNCTIONING AS A TEAM

On power up all the registers are reset, ensuring the display's shutdown. The five control registers handle the following functions:

Register C enables the chip shutdown. When the data is 0, all scanning ceases, and the segment current is shut off. You can update any register while the chip is in shutdown mode. You can use this register to flash the display on and off simply by writing alternating ones and zeros.

Register B controls the scan limit. You may include any number of digits in the multiplexed scan; generally it is set once and forgotten because it should match the number of display digits installed.

Register A controls the intensity. A resistor from ISET to Vcc sets overall segment current to 100 times ISET. Typical values would be 10k for about a 40-mA segment current; however, this value will depend on the actual LED drop. For this reason you may wish to use 8.2k and a 50k trimmer. PWM drives each segment. The value in the intensity register will determine the ratio of the PWM.

Register F enables the display test. This test turns on all segments at maximum intensity.

Register 0 is for NOP. However, this register is not real; when written to, no action takes place. Sound silly? Here is why you might need it. Say you're using a daisy chain of displays, but wish to update only one display; NOPs to all the others in the chain will ensure that those displays remain unaffected when a load is performed.

Register 9 controls the digit decode. Each bit position O-7 within the register relates to the decode mode of the individual digit O-7. A zero in a bit position means the data for that digit is in the form of one bit for each segment. A one means the data for that digit is in a decoded mode-a zero means digit 0, a one means digit 1, and so forth. You might think that in decoded mode, data values O-15 translate to digits O-9 and A-F, but they don't. Instead, the supported characters are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, –, H, E, L, P, and blank. In either mode, bit 7 sets the decimal point.

Registers 1-8 are the digit registers. These RAM locations hold the data for each of the eight digits. The data is interpreted differently depending on the data in register 9—the digit decode. For example, if register 1 (digit 0) is a one and bit 0 of register 9 is a zero, then a single segment (segment G) is on. If bit 0 of register 9 is a one, then two segments will be on: the segments for a numeral 1.

READY, AIM, CLOCK

Data is sent to the '7219 in 16-bit words (MSB to LSB). When multiple displays are used, the data-out line force-feeds bits to the next module, daisy-chain style. Only the clock and load lines are common to all modules. As the number of modules in a chain increases, the number of 16-bit data words needed expands as well. The data must fill the shift registers of all '7219s and be loaded into all modules at once.

The circuit layout should be 4" wide (if you choose to use the same display as I have) to allow the addition of the modules end to end. Keep the segment and digit lead away from the clock, data, and especially the load line. Coupled spikes could cause an unwanted load. I used DO, D1, and D2 (pins 2, 3, and 4) along with ground (25) on my PC's parallel port to provide

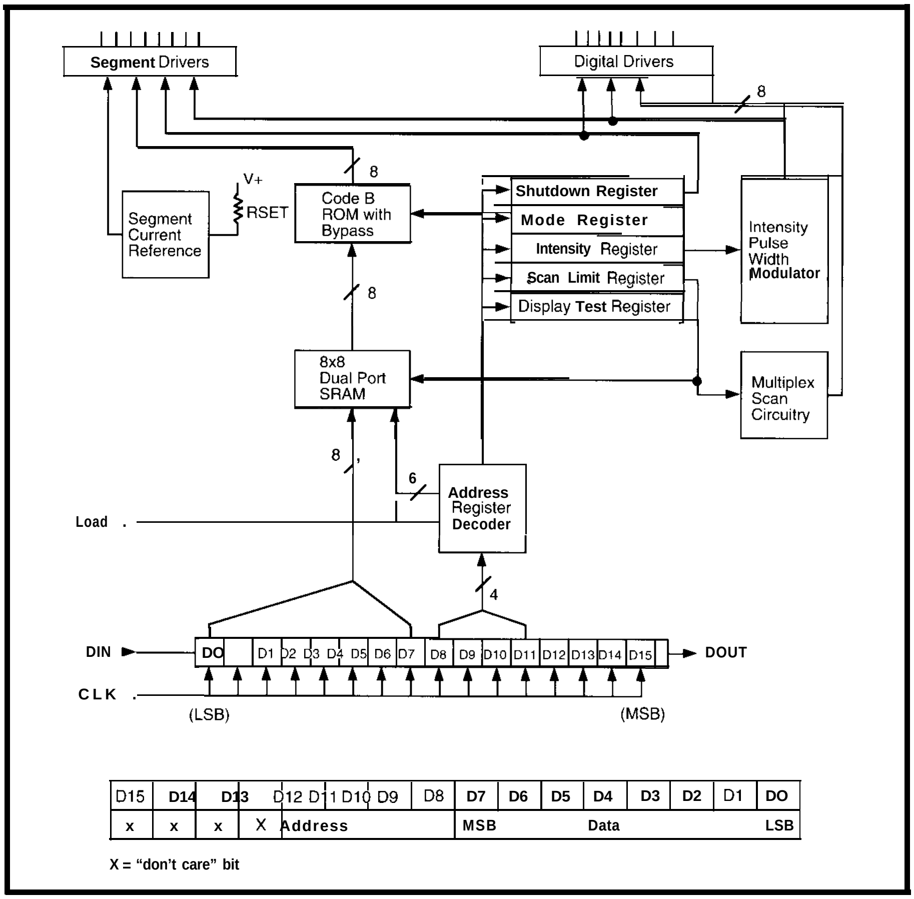


Figure 2—*Part* of *what makes up the MAX7219 is ifs 16-bit shift register. The four most-significant cits are nor used. However, bits DO to D7 serve as the real data used while bits D8 to D11 are used for addressing.*

Listing 1—*Control of the MAX7219 through the PC's parallel port can even be done from interpreted BASIC.*

```
10 HCLK = 1: LCLK = 0
20 HDIN = 2: LDIN = 0
30 HLOAD = 4: LLOAD = 0
40 V = 0
50 DPORT = &H378
60 OUT DPORT,&HO
70 DECODE = 0: INTENSITY = 0: SCANLIM = 0: SHUTDOWN = 0
80 DISPTST = 0 :CHARACTER = 0
130 PRINT "SELECT A ITEM"
140 PRINT: PRINT "1 SEND A WORD WITHOUT A LOAD"
150 PRINT "2 SEND A WORD WITH A LOAD"
160 PRINT "3 BUILD A WORD AND SEND WITHOUT A LOAD"
170 PRINT "4 BUILD A WORD AND SEND WITH A LOAD"
180 PRINT "5 QUIT"
190 I$ = INKEY$
200 IF I$="" THEN 190
210 IF I$="1" THEN LOD = LLOAD: GOT0 270
220 IF I$="2" THEN LOD = HLOAD: GOT0 270
230 IF I$="3" THEN LOD = LLOAD: GOT0 480
240 IF I$="4" THEN LOD = HLOAD: GOT0 480
250 IF I$="5" THEN STOP
260 GOT0 190
270 INPUT "ENTER THE WORD (O-65535)". W
280 FOR Z=15 TO 0 STEP -1
290 PRINT Z;
300 IF (W<2^Z) THEN V = (V AND &HFD) ELSE V = (V AND &HFD) OR 2
310 IF (W>=2^Z) THEN W = W-2^Z
320 OUT DPORT,V
330 PRINT V;
340 V = (V AND &HFE) OR HCLK
```

(continued)

Listing I-continued

```
350 OUT DPORT,V
360 PRINT V;
370 IF Z=O THEN V = (V AND &HFB) OR LOD
380 OUT DPORT,V
390 PRINT V:
400 V = (V AND &HFE) OR LCLK
410 OUT DPORT,V
420 PRINT V;
430 IF Z=O THEN V = (V AND &HFB)
440 OUT DPORT,V
450 PRINT V
460 NEXT Z
465 PRINT "I=";I
470 GOT0 130
480 PRINT
490 PRINT "SELECT OPTION"
500 PRINT: PRINT "0 NO-OP"
510 PRINT "1 DIGIT 0"
520 PRINT "2 DIGIT 1"
530 PRINT "3 DIGIT 2"
540 PRINT "4 DIGIT 3"
550 PRINT "5 DIGIT 4"
560 PRINT "6 DIGIT 5"
570 PRINT "7 DIGIT 6"
580 PRINT "8 DIGIT 7"
590 PRINT "9 MODE (BCD/SEGMENT)"
600 PRINT "A INTENSITY"
610 PRINT "B SCAN LIMIT"
620 PRINT "C SHUTDOWN"
630 PRINT "F DISPLAY TEST"
640 I$ = INKEY$
650 IF I$="" THEN 640
660 IF (I$>"/" AND I$<":") THEN I = ASC(I$)-&H30: GOT0 690
670 IF (I$>"@" AND I$<"G") THEN I = ASC(I$)-&H37: GOT0 690
680 GOT0 640
690 IF (I=13 OR I=14) THEN GOT0 640
700 IF (I>0 AND I<9) THEN GOT0 780
710 IF (I=9) THEN GOT0 1000
720 IF (I=10) THEN GOT0 1120
730 IF (I=11) THEN GOT0 1240
740 IF (I=12) THEN GOT0 1360
750 IF (I=15) THEN GOT0 1490
760 W = I
770 GOT0 280
780 IF (DECODE AND 2^(I-1))<>0 THEN GOT0 840
790 PRINT
800 PRINT "BIT POSITION SEGMENT (0=OFF,1=ON) = CHARACTER"
810 PRINT "D7 D6  D5  D4  D3  D2  D1  DO"
820  PRINT "DP  A   B   C   D   E   F   G"
830 GOT0 880
840 PRINT
850 PRINT "DISPLAY = CHARACTER"
860  PRINT "0=0, 1=1, 2=2, 3=3, 4=4, 5=5, 6=6, 7=7, 8=8, 9=9,"
870 PRINT "-=10, E=11, H=12, L=13, P=14, blank=15, DP=add 128"
880 PRINT
890 PRINT "THE CURRENT CHARACTER FOR DIGIT";I-1;" IS": CHARACTER
900 PRINT "SELECT OPTION"
910 PRINT: PRINT "1 CHANGE CHARACTER"
920 PRINT "2 SEND IT"
930 PRINT "3 CANCEL FUNCTION"
940 I$ = INKEY$
950 IF I$="" THEN 940
960 IF (I$="3") THEN GOT0 480
970 IF (I$="2") THEN I=(I*256)+CHARACTER: GOT0 760
980 IF (I$="1") THEN INPUT "ENTER THE NEW CHARACTER (0-255)",
     CHARACTER : CHARACTER = CHARACTER AND &HFF
990 GOT0 780
```

(continued)

Listing 1-continued

```
1000 PRINT
1010 PRINT "THE DECODE MODE IS CURRENTLY "; DECODE
1020 PRINT "SELECT OPTION"
1030 PRINT: PRINT "1 CHANGE DECODE MODE"
1040 PRINT "2 SEND IT"
1050 PRINT "3 CANCEL FUNCTION"
1060 I$ = INKEY$
1070 IF I$="" THEN 1060
1080 IF (I$="3") THEN GOT0 480
1090 IF (I$="2") THEN I = (I*256)+DECODE: GOT0 760
1100 IF (I$="1") THEN INPUT "ENTER THE NEW DECODE MODE (0-
      255)", DECODE: DECODE = DECODE AND &HFF
1110 GOT0 1000
1120 PRINT
1130 PRINT "THE DUTY CYCLE IS CURRENTLY "; INTENSITY
1140 PRINT "SELECT OPTION"
1150 PRINT: PRINT "1 CHANGE DUTY CYCLE"
1160 PRINT "2 SEND IT"
1170 PRINT "3 CANCEL FUNCTION"
1180 I$ = INKEY$
1190 IF I$="" THEN 1180
1200 IF (I$="3") THEN GOT0 480
1210 IF (I$="2") THEN I = (I*256)+INTENSITY: GOT0 760
1220 IF (I$="1") THEN INPUT "ENTER THE NEW DUTY CYCLE (0min-
      15max)", INTENSITY : INTENSITY = INTENSITY AND &HF
1230 GOT0 11'20
1240 PRINT
1250 PRINT "THE NUMBER OF ACTIVE DIGITS IS CURRENTLY "; SCANLIM+1
1260 PRINT "SELECT OPTION"
1270 PRINT: PRINT "1 CHANGE NUMBER OF DIGITS"
1280 PRINT "2 SEND IT"
1290 PRINT "3 CANCEL FUNCTION"
1300 I$ = INKEY$
1310 IF I$="" THEN 1300
1320 IF (I$="3") THEN GOT0 480
1330 IF (I$="2") THEN I = (I*256)+SCANLIM: GOT0 760
1340 IF (I$="1") THEN INPUT"ENTER THE NEW NUMBER OF ACTIVE
      DIGITS (1-8)", SCANLIM: SCANLIM=SCANLIM-1: SCANLIM =
      SCANLIM AND &H7
1350 GOT0 1240
1360 PRINT
1370 PRINT "THE SHUTDOWN MODE IS CURRENTLY ";
1380 IF (SHUTDOWN=1) THEN PRINT "OFF" ELSE PRINT "ON"
1390 PRINT "SELECT OPTION"
1400 PRINT: PRINT "1 TOGGLE SHUTDOWN MODE"
1410 PRINT "2 SEND IT"
1420 PRINT "3 CANCEL FUNCTION"
1430 I$ = INKEY$
1440 IF I$="" THEN 1430
1450 IF (I$="3") THEN GOT0 480
1460 IF (I$="2") THEN I = (I*256)+SHUTDOWN: GOT0 760
1470 IF (I$="1") THEN IF (SHUTDOWN=0) THEN SHUTDOWN = 1 ELSE
      SHUTDOWN = 0
1480 GOT0 1360
1490 PRINT "THE DISPLAY MODE IS CURRENTLY 'I;
1500 IF (DISPTST=0) THEN PRINT "OFF" ELSE PRINT "ON"
1510 PRINT "SELECT OPTION"
1520 PRINT: PRINT "1 TOGGLE DISPLAY MODE"
1530 PRINT "2 SEND IT"
1540 PRINT "3 CANCEL FUNCTION"
1550 I$ = INKEY$
1560 IF I$="" THEN 1550
1570 IF (I$="3") THEN GOT0 480
1580 IF (I$="2") THEN I = (I*256)+DISPTST: GOT0 760
1590 IF (I$="1") THEN IF (DISPTST=0) THEN DISPTST = 1 ELSE
      DISPTST = 0
1600 GOT0 1490
```

the necessary control of the seven-digit display. Use Figure 3 as a guide if you wish to write code for your favorite micro. I wrote Listing 1 in GWBASIC, and it will give you complete control of every register of the '7219 through your PC's parallel port.

BEWARE, ICEBERGS AHEAD

When I choose to use a specific IC to help tackle a task, I do so because I enjoy the smell of solder more than the sound of my fingers tapping the keyboard in programming mode. Although the '7219 is a single-chip solution to supporting seven-segment displays, it does require three routines. The MAX7219 must be initialized, the digit information must be coded into a 16-bit word, and the data must be bit banged through three output bits.

Data throughput is limited to 100 ns per bit. Even at 16 bits, or 1.6 us, you can see that the '7219 is no slouch. Most controllers will be unable to achieve to these levels, so there can be no complaint about speed.

After 16 bits have been shifted by the clock line, the word is put in the appropriate register when the load line is raised.

DISPLAYS ON PARADE

There are all sizes of single- and multidigit seven-segment displays. The smallest, used in the first calculators, had built-in lenses to magnify the bitty numerals. Although 3" digits (and larger) are available, prepare to spend $10 or more per digit. Because these larger displays' digits are made up of multiple LEDs per segment, they use multiple junction drops. I believe today's best buys are the one- and two-digit displays that have a numeral height of 0.6". Not only are these readable from across the room, but they have a cost per digit of about a buck in small quantities. Every display manufacturer I've looked at has a cross for these HDSP5503 displays and they come in red, green, and yellow.

LED displays other than seven-segment can be used with the '7219. An 8 x 8 matrix character display could display a single character or eight vertical or horizontal bar graphs. However, these displays can be costly,

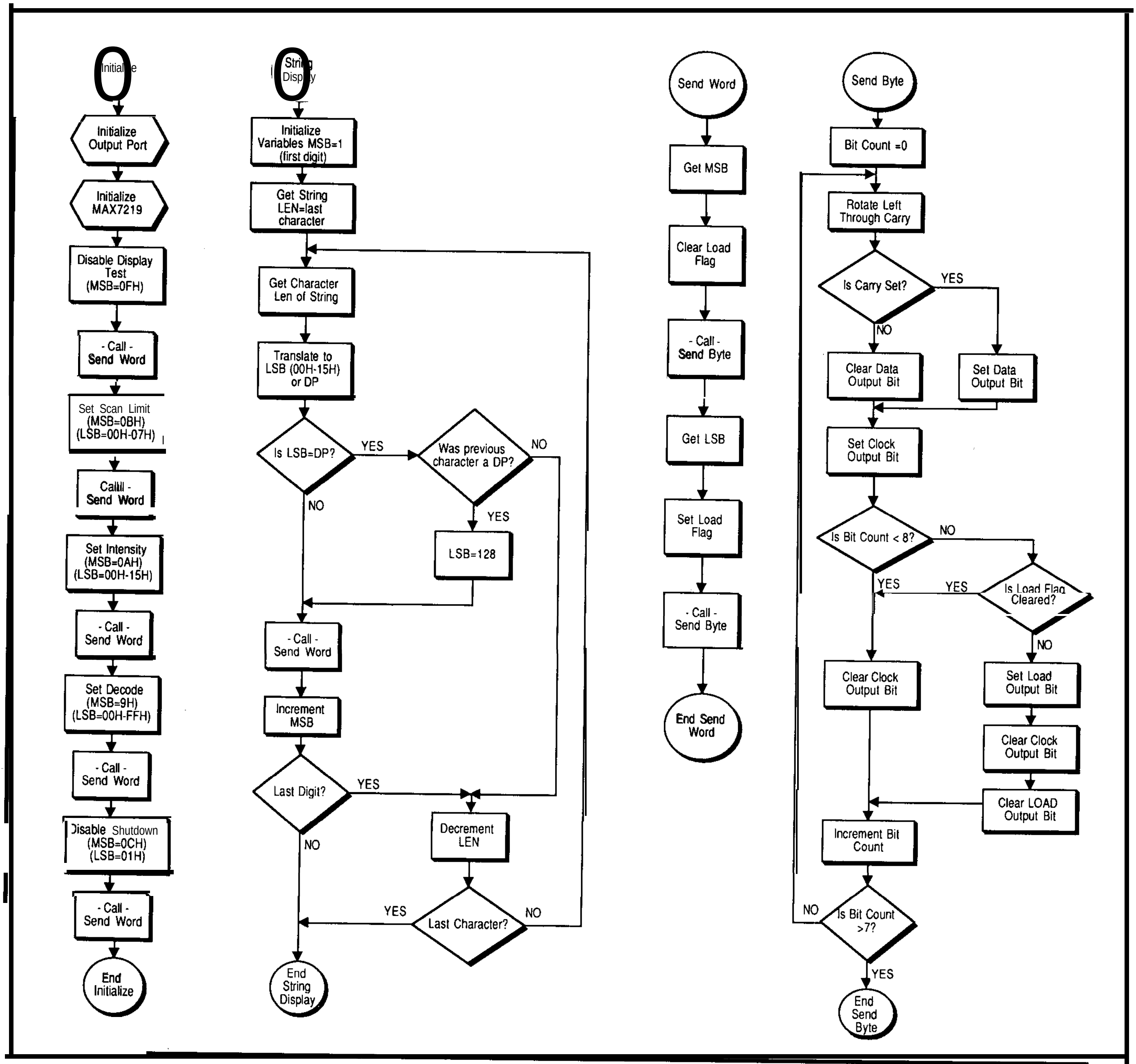


Figure 3—*While the MAX7219 makes the hardware interface simple, it does require extra software to shift the data out of the host computer one bit at a time.*

and multiple characters require multiple drive chips.

NUTSHELL

Combining bits of circuitry into a specialized IC will simplify design and increase reliability of any manufactured product. The designer often reaps the further benefit of reduced power usage. Maxim's '7219 really cuts the external parts count to the bone, requiring only one external resistor to set the display's segment current.

System interfacing is reduced to three control lines. A clock and data line shift information into the IC while a separate load control grabs 16 bits at a time from the pipeline.

Simplicity, lower power, and the capability of remote operation, increase the potential use of this innovative IC in your next design. After all, could you design a low-parts-count eight-digit display module for around $15? ❑

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*** *manufac*-***turing.***

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

68k The Easy Way

Motorola hits the marl. with the 68306

As today's embedded applications tax 8-bit microcontrollers' capabilities to the limit, the search continues for suitable 16/32-bit replacements. Motorola's latest offering should fill the bill in many cases.

SILICON UPDATE

Tom Cantrell

No doubt 8-bit chips such as the 8051, 68HC11, and Z180 will remain the high-volume embedded control workhorses for the future. Indeed, shrinking die and savvy marketing have yielded very inexpensive variants that compete with 4-bit chips in even the most miserly of applications.

That's the good news. The bad news is at the other end of the spectrum, the place where 8-bit chips start to run out of gas. A particular problem is the 64K address space limit, which the move to C exacerbates. Try to cram a 10,000-line C program onto an 8-bit chip, especially one using floating point, and you'll see what I mean.

Sure, 8-bit chips can adopt bank-select or other segmentation schemes to dance around the 64K problem, but why bother? If it weren't for the PC's blessing of the concept, I'm sure segmentation would have been relegated long ago to the ash heap of computing history where it belongs.

A 32-bit address space gives the breathing room that big programs and data sets require. The question is how to make the 64 KB-to-4 GB transition, while avoiding 32-bit sticker shock.

IF YOU BUILD IT, THEY WILL COME

I'm surprised Motorola took this long to come up with the idea of 68k-based embedded controllers. Now

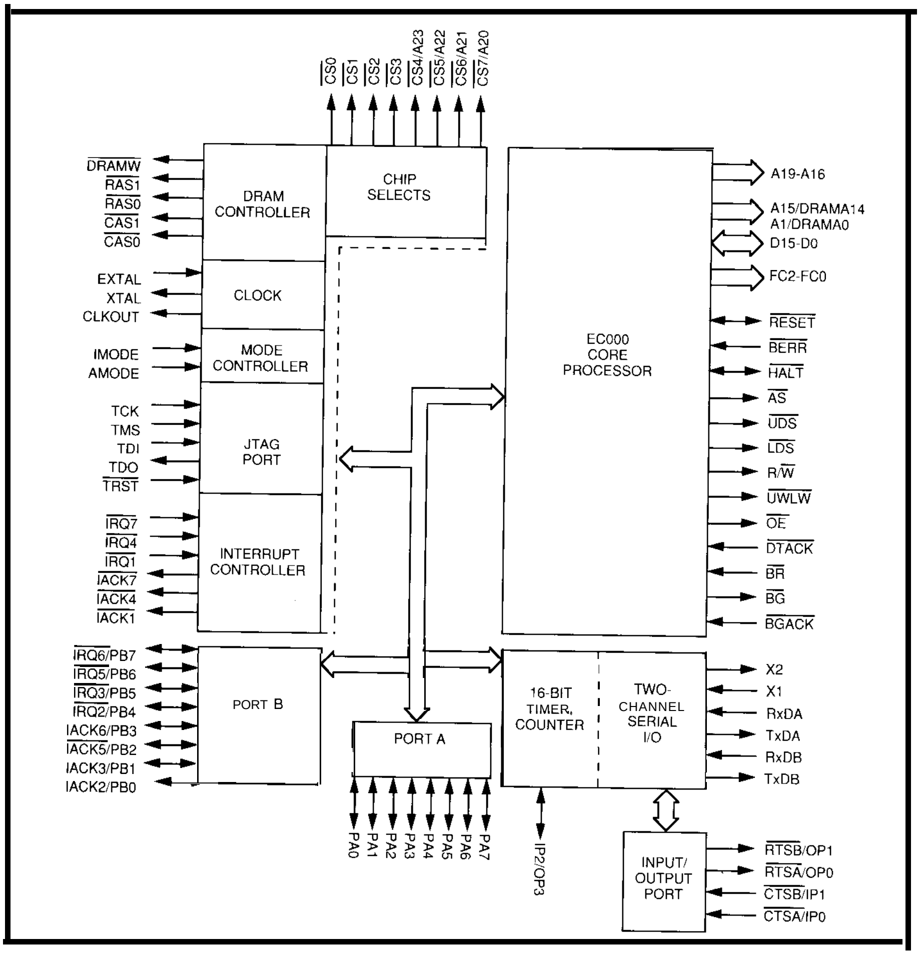


Figure 1—The MC68306's multitude or I/O pins and interrupts, ifs direct support of DRAM, and many other on-chip features make it a contender to replace traditional 8-bit parts in many applications.

they've definitely seen the light, and the 683xx family is the result (see Photo 1).

Motorola defined their first 683xx chips in response to specific customers and applications; thus, these chips tend to offer specialized features unnecessary in most applications. Consider the MC68332: designed as an automotive engine control unit (ECU) for GM, it contains a powerful Time Processing Unit. It's grand if you need to generate eight cylinders worth of spark, fuel injection, and other engine timing, but it is overkill otherwise.

Similarly, the MC68F333, the first 32-bit single-chip with on-chip flash memory rather than the conventional ROM or EPROM, was originally designed as an ECU for BMW. Of course, such specialization comes at a price; even after a recent reduction, the 'F333 is close to $100.

Another example is the MC68302, which includes three powerful X.25-type SIOs. It's a perfect fit for ISDN applications, but way beyond the typical application's need for a simple UART or two.

Worthy as these parts are in their niche applications, they aren't a likely solution for the middle-of-the-road 8-bit designer who just needs a simple, inexpensive way to upgrade to 32 bits.

Finally, almost 15 years since the first 68000 rolled off the fab lines, Motorola has come up with the MC68306, a part that can compete with S-bit chips for low-cost, high-volume applications.

TAKING CARE OF "BITNESS"

A quick glance at the '306 (see Figure 1) shows that it combines a 68EC000 core CPU, a 6868 1 DUART, some parallel I/O, and the glue logic required to cut chip count and system cost.

You may call to task my assertion that the '306 is a "32-bit" CPU if you notice the physical data bus (DO-D15) is only 16 bits. So before describing the '306 further, let me give you my bit about bits.

CPU	Ext. Bus Width	ALU Width	Programmers Model
8051	8	8	16
Z80/180	8	8	16
80196	16	16	16
6805	8	8	16
68HC11	8	8	16
68HC16	16	16	16
8088	8	16	16
86/186/286	16	16	16
386SX	16	32	32
386DX	32	32	32
68008	8	16	32
68000	16	16	32
68306	16	16	32
6833x	16	32	32
68020/30/40	32	32	32

Table 1-*One measure that can be used to compare processors is their "bitness," which can be determined by looking at data bus width, ALU width, and, most important/y, the width of the 'programmer's mode/. "*

Generally, the "bitness" of processors, in the absence of a strict measurement criterion, has been subject to abuse by marketing departments. The situation is similar to that for benchmarks, which can be misleading and subject to marketing-driven mischief despite noble efforts like SPECmark.

The potential for confusion arises because there are many places to measure the bitness. To see through the hype, I've found characterizing three levels (there are more if you're picky) for any given CPU useful.

The easiest measure is the width of the physical data bus (i.e., just count the *D* pins). As I've mentioned, by this criterion the '306 is a "16-bit" chip.

At the inner core of every CPU chip is an Arithmetic Logic Unit (ALU) that performs the basic arithmetic (ADD, SUB, etc.) and logic (AND, OR, etc.) operations as its name implies. The 'EC000 core uses a 16-bit ALU as the 68000 did before it.

Finally, there's the "programmer's model," which I interpret as a combination of the register and ALU width as perceived by the instruction set. This point is where I get into trouble with fans of paging and segmentation (believe me, some will write) and where I get the nerve to classify the '306 as a "32-bit" chip. The '306 registers are all a full 32 bits wide and, furthermore, the instruction set accepts 32-bit commands (i.e., arithmetic and logic operations on 32-bit operands), even though the 16-bit ALU executes them in "Texas Two-step" fashion.

The number of pins and the width of the ALU circuit are implementation issues that can migrate gracefully with technology. Indeed, a '306 user has a copious upgrade path to 32-bit ALUs (e.g., MC68340) and buses (e.g., 68EC020) within the perpetually expanding 68k lineup.

Thus, I feel the programmer's model criterion is most important because it imposes fundamental architectural limits. Unlike adding pins or ALU transistors, changing a 16-bit programmer's model to a 32-bit programmer's model is very hard to do. In fact, you make this change only by adding a new mode for new code, while keeping the old mode for old code as PC programmers grappling with real (≤286) versus protected (2386) mode know well.

Table 1 summarizes the three measures of bitness for a variety of popular control CPUs. As you can see, segmented/paged machines, such as the '180, 'HC16, and '86/'186/'286, only rate 16 bits for their programmer's models, though all can address more than 64 KB of memory with varying degrees of blood, sweat, and tears.

I say if you're chronically bumping into the 64-KB problem, now is the time to wake up, smell the coffee, and make the big move to a chip with a 32-bit programmer's model like the '306.

HARD "CELL" TACTICS

Perhaps the rather convoluted path to the '306 was the result of the "SICness" that plagued the IC industry in the eighties.

First came Application Specific IC (ASIC), a concept that I've rather disparaged in the past. In it, customers are supposed to whip out a chip design themselves, send it off to a foundry for fabrication, and live happily ever after.

However, large segments of the market (especially smaller customers) have shied away from accepting both the technological (i.e., getting the chip working] and inventorial (eat the parts if your widget doesn't sell] risks. Indeed, the legions of ASIC-don't-wannabes have been the ones fueling the growth in field-programmable logic such as PALs,PLDs, FPGAs, and so forth. The proponents of ASIC failed to recognize good design technology (i.e., gate array or standard cell) rather than a product.

Value of	At RAS When DRSZ(2:0) Is:								At CAS:
	111	110	101	100	011	010	001	000	
DRAMA14	A30	A29	A28	A27	A26	A25	A24	A23	A15
DRAMA13	A29	A28	A27	A26	A25	A24	A23	A22	A14
DRAMA12	A28	A27	A26	A25	A24	A23	A22	A21	A13
DRAMA11	A27	A26	A25	A24	A23	A22	A21	A20	A12
DRAMA10	A26	A25	A24	A23	A22	A21	A20	A19	A11
DRAMA9	A25	A24	A23	A22	A21	A20	A19	A18	A10
DRAMA8	A24	A23	A22	A21	A20	A19	A18	A17	A9
DRAMA7	A23	A22	A21	A20	A19	A18	A17	A16	A8
DRAMA6	A22	A21	A20	A19	A18	A17	A16	A15	A7
DRAMA5	A21	A20	A19	A18	A17	A16	A15	A14	A6
DRAMA4	A20	A19	A18	A17	A16	A15	A14	A13	A5
DRAMA3	A19	A18	A17	A16	A15	A14	A13	A12	A4
DRAMA2	A18	A17	A16	A15	A14	A13	A12	A11	A3
DRAMA1	A17	A16	A15	A14	A13	A12	A11	A10	A2
DRAMA0	A16	A15	A14	A13	A12	A11	A10	A9	A1

'able *P-The real breakthrough for the '306 is the on-chip address multiplexers. Lines A15–A1 serve double duty as* DRAM *muxed addresses.*

Motorola came up with the smarter, if questionably named, Customer Specific IC (CSIC) approach in which they design a special chip at the request of a customer. This method centralizes the IC design function with the experts at Motorola as opposed to the ASIC hope of teaching everyone to be a chip designer. What differentiates CSIC from the traditional custom approach is Motorola's offer to absorb the infamous Non-Recurring Engineering (NRE) cost in return for the right to sell the "custom" design to other customers as a standard product. NRE is what really starts to hurt ASIC customers after the usual multiple fab go-arounds needed to get a chip working. The CSIC approach explains the customer-specific (i.e., GM, BMW, etc.) nature of the early members of the 683*xx* family.

The latest trend in chip design is the *standard cell approach* in which standard silicon function blocks are stitched together on a single chip much as chips are traditionally combined on a board. Interestingly, in their "Motorola CSIC Design Methodology" documents from a few years back, Motorola disparages the standard cell approach as a "notoriously inefficient...shake-and-bake" scheme.

Much as political campaign promises evaporate in the harsh light of post-election reality, it turns out that Motorola now recognizes that the standard cell approach is actually a great way to spin a lot of different parts with quick and easy leveraging of existing design know-how. Furthermore, fundamental market forces—namely, that silicon is always cheaper while design time (i.e., people) is ever more expensive-are inexorably marching on. Thus, the '306 is the first to use what they proudly proclaim as the "company's new standard cell design methodology."

So what if Motorola has to eat a little shook-and-baked crow, at least now I think they've finally got it right.

I HAVEN'T GOT A GLUE

I'm going to dispatch with describing more than half the '306 die, the 'EC000 core CPU, and the 68681 DUART rather quickly for a couple of reasons. First, they are simply straightforward "cell" implementations of well-known standard parts. Second, the remaining glue logic (DRAM and interrupt controllers, chip selects, etc.) is what really sets the '306 apart from its predecessors and positions the chip for contention in the S-bit arena.

The 'EC000 core is basically a 68000 CPU. The main difference between the '306 'EC000 core and the 68000 is the deletion of the latter's 6800-peripheral interface feature. That isn't really a problem because modern replacements for the old 6821 PIA, 6840 PTM, and 6850 ACIA chips are available.

The DUART cell is also quite similar to the stand-alone device, combining two independent UARTs [each with its own 50- to 38.4k-bps generator), a 1 6-bit timer, and some general-purpose I/O lines. Where the MC6868 1 offers a total of 14 I/O lines, the '306 only brings out five (two out, two in, one out or in). The inputs feature change-of-state detectors, supporting their use as modem control inputs (CTS, DCD, etc.). The UARTs also support the popular 9-bit mode for multidrop connection between '306s or other CPUs (e.g., 8051, 2180, 68HCll) with that feature.

Note that the DUART cell calls for its own clock inputs via crystal or CMOS level on the Xl and X2 pins in addition to the main CPU clock (EXTAL and XTAL). I guess this requirement is an example of standard cell inefficiency, but it hardly seems notoriously so.

Oh, there is also an 8-bit parallel I/O port (PORT A). This feature is a major boon for embedded designers who, in their battle to maximize function and minimize cost, often proclaim, "My kingdom for an I/O line."

Now let me get to the good stuff: the glue logic that makes minimalist '306 designs possible and, indeed, even easier than for 8-bit chips.

First is the interrupt controller. The 68000 defined an eight-level priority interrupt scheme, which the '306 carries forward, but where the 68k encoded the interrupt level on three lines, the '306 puts the encoder on chip. Needing an extra interrupt ranks right up there with needing an extra I/O line, and the '306's seven input request lines put it way ahead of older CPUs, which usually offer only two or three.

The '306 allows the active level (high or low) of each interrupt to be

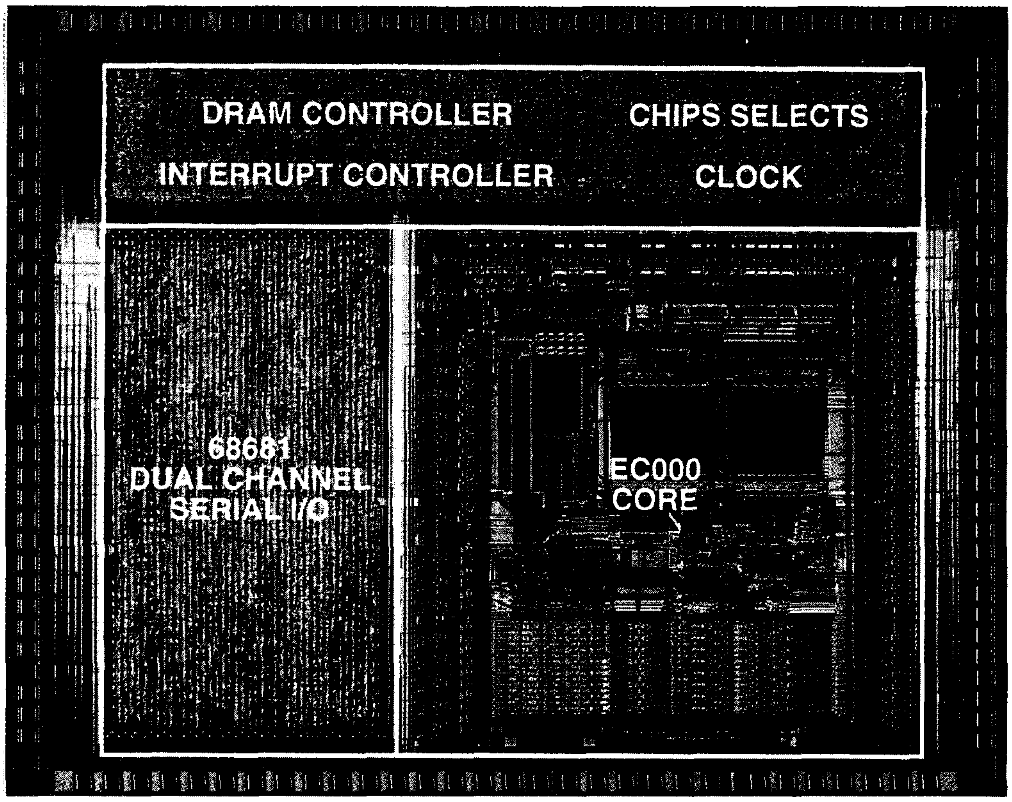


Photo 1—*Motorola does the ASIC concept one betterwith ifs CSIC (customer-specific integrated circuit). Using a standard processor core, Motorola engineers design the chip to a customer's specifications with the understanding that the final chip may be offered for sale to others.*

defined independently, eliminating the need for an external inverter. Also, you can define each input to generate a vectored or an autovectored response. The latter sends an interrupt to a fixed location and the former asks the interrupting device (using an IACK* pin) where to go.

Notice how the IRQn*/IACKn . lines are split into a dedicated portion (n = 1,4,7} and a set (n = 2,3,5,6} that can be configured as even more I/O lines (PORT B).

Next is the chip select controller that offers up to eight CS* lines. Given the high density of today's peripheral and memory ICs, eight chip selects are more than enough to support rather large systems. Indeed, the '306 offers a natural tradeoff, allowing programmable definition of four lines as either CS* or the most-significant address bits, A23–A20. Because each CS* line can access up to 16 MB within the 4-GB space, the number of address lines doesn't really limit memory expansion.

As in many features, the 68k and Intel families defined essentially different mechanisms for timing bus cycles. The 'x86 used a synchronous READY scheme in which bus logic must actively request a wait state. The 68k relied on asynchronous DTACK*, which is kind of the inverse; external logic must actively acknowledge a data transfer for it to complete.

Without rehashing the old religious wars concerning the issue, just note that in the DTACK' scheme, every system-even one with O-wait states-needs DTACK* generation logic. Furthermore, should that logic fail to respond (e.g., if the CPU attempts to access a nonexistent I/O chip], the CPU will simply hang in an infinite wait state.

The '306 addresses both issues by including DTACK* generation logic with the chip select controller, allowing the access time for each CS* region to be set independently. Motorola recognized the worst case combination of fast CPU and slow peripheral by providing up to 950 ns (at 16.67 MHz) of access time. Should you want to use an external DTACK' (say for a really slow chip], they provide a bus watchdog timer that automatically terminates an access should it remain stuck longer than a specified time (up to 4096 clocks).

Another neat feature is the capability to specify if reads, writes, or both can take place for a given CS . Also, because the 68k architecture supports USER/SUPERVISOR protection, you can limit a CS . to supervisor-only access. Together these protections provide an extra measure of safety in hazardous or high-reliability applications.

Last but not least is the DRAM controller, which is one of my favorite features of the '306.

TAKE IT TO THE MUX

The '306 isn't the first chip to include some DRAM support. In fact, one of the original claims to fame for the venerable Z80 was its stab at supporting DRAMs. Nevertheless, connecting DRAMs has always required a degree of outside logic whether it be refresh control, RAS . /CAS * generation, address multiplexing, or so forth.

Maybe everyone was waiting for the memory guys to wake up and make a micro-oriented, not mainframe-oriented, DRAM. For example, why are the lowest price/bit DRAMs always the xl variety when that causes a severe granularity problem for even a x8 memory bank, not to mention xl6 or x32? Memory suppliers' obstinacy has led to the SIMM concept, an expedient way to make a x8, x9, or even x36 "DRAM" using commodity xl or x4 chips. One interesting side effect: the recently bankrupt Wang turns out to have a patent for the SIMM concept for which they now collect royalties. So, "Save a lawyer, buy a SIMM."

I guess the micro folks have finally given up on the DRAM designers; Motorola is one of the first (they won't be the last) to bite the bullet and put a complete DRAM interface on the CPU. When I say "complete" I mean you can connect plain DRAMs to the '306 with absolutely zero extra logic!

For refresh timing, the '306 includes a timer and bus arbitration logic that will trigger a refresh as needed. Note that it uses a form of refresh called *CAS-before-RAS,* which is distinguished by not needing a refresh address.

CAS-before-RAS refresh is especially suitable because the '306 itself generates two banks worth of RAS* and CAS* signals. Because CAS-before-RAS refresh doesn't need an address, refreshing the DRAMs simultaneously with other bus operations such as EPROM or I/O access is possible.

The real breakthrough is on-chip address multiplexers. You'll notice that the A15–A1 lines serve double duty as DRAM muxed addresses DRAMA14–DRAMA0. As shown in Table 2, a "DRAM Size" field (DRSZ) in the '306 assigns RAS*/CAS* address outputs to the CPU pins in order to accommodate various DRAM sizes. Feasible no-glue DRAM systems encompass everything from 128 KB (the minimum bank size) using a measly four 64K x 4 (256-kilobit) DRAMs all the way to 16 MB using 32 4M x 1 DRAMs. In fact, the '306 scheme is ready for 16- and even 64-megabit DRAMs when they reach production status.

The bottom line is a '306 system can do more than most other popular control CPUs, and do it with fewer chips, too. Consider the tidy design shown in Figure 2 that provides 256K of EPROM and 512K of DRAM, not to mention the UARTs, timer, parallel I/O, and so forth. Because the design easily scales using higher-density memories, "eight (chips) is enough" when it comes to '306 designs is something safe to say.

PARITY ENABLED

The technical features of the '306 are nice, but they aren't what really sets it apart.

Until now, breaking the 32-bit barrier could also break your budget. Besides a more complex and chip-intensive design, there was the simple matter of double-digit price tags for the CPU.

Remarkably, the '306 nearly achieves price parity with top-end 8-bit chips and shatters the double-digit price barrier at only $8.95 (10,000). Furthermore, the few dollars premium over a $5-class 8-bit chip is easily offset once you fully utilize '306 glue features.

Due to the long wait for the chip, the '306 isn't the only "beyond 64K" solution (stay tuned.. .), but it's a darned good one. ❑

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

CONTACT

Sue Cozart
Motorola, Inc.
6501 William Cannon Dr. W.
MD:OE33
Austin, TX 78735
(512) 891-2134
Fax: (512) 891-2943

IRS

419 Very Useful
420 Moderately Useful
421 Not Useful

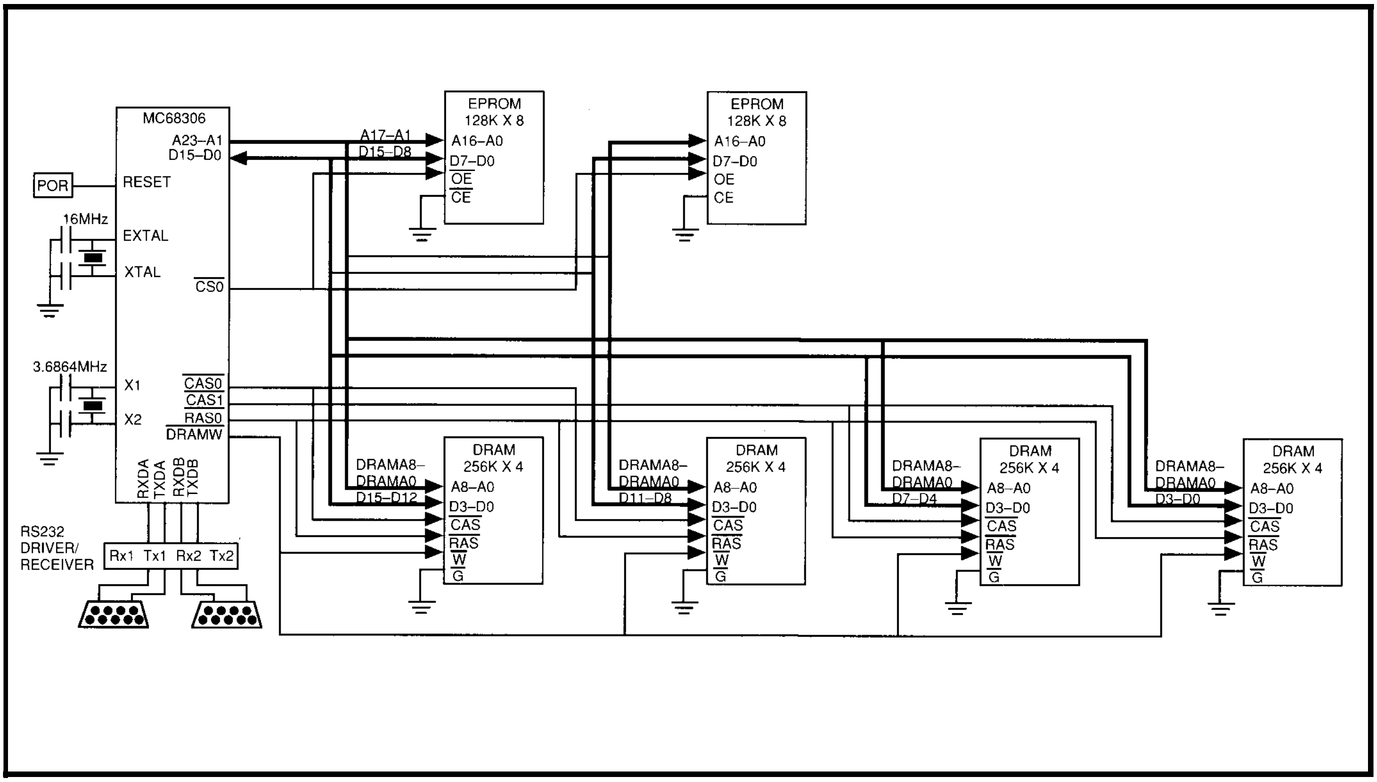


Figure **2**—*In a sample system with 256K of EPROM and 512K of DRAM, the 68306 requires virtual/y no external glue chips, making it an idea/ microcontroller in situations where 8 bits just won't do.*

Displays in a Vacuum

Using vacuum fluorescent displays

While LED and LCD displays are usually used to convey information to the user, vacuum fluorescent displays still have a place in applications where high visibility is important. FInd out the basics of using VFDs.

EMBEDDED TECHNIQUES

John Dybowski

Some controllers exist in apparent isolation and carry out their solitary missions invisibly. However, at times you have to carry communications to the outside world using some visible means. The complexity of the visual interface can range from simple incandescent bulbs or LEDs to intricate character-based or graphic readouts using LCDs, CRTs, or other display devices.

For many controller projects, the output devices can be nothing more complicated than a couple of LEDs. Generally, as the system's capacity for processing data grows, the need for sophisticated output devices becomes more of a necessity. If you're faced with a requirement for a character-based output device, many different options exist, and you may find yourself considering which display technology to buy for inclusion in your new system. Probably the first thing that comes to mind is using one of the many LCD devices available.

Due to their low cost and low power requirements, LCD panels are particularly popular these days. Previous problems have been corrected to a great extent, and a number of technologies address the need for backlighting. Electroluminescent panels have been used in this regard for some time now, but suffer from a limited operating life of about 10,000 hours and the need of a power inverter. LED backlighting has a much longer life span and is gaining popularity, but power consumption becomes a real problem, especially when used with larger LCD panels. Perhaps the biggest drawback with LCDs is their limited temperature range, restricting their usefulness in outdoor applications.

Gas discharge tubes also find many uses in indicator panels. These are the familiar AC and DC plasma displays popular on many panel meters and various types of test equipment. The displays require firing voltages in excess of 100 volts to ionize the gas and maintain conduction.

AC thin film electroluminescent displays are solid-state devices featuring a compact size and low power requirements. When the alternating voltage applied across two crossing electrodes exceeds the threshold, the luminescent layer emits light.

Vacuum fluorescent displays (VFDs) are available in a variety of character and dot matrix configurations. All but the smallest configurations use dynamic drive methods in order to minimize the number of required drivers and the number of panel pinouts. Due to the relaxed dynamic timing constraints and the relatively low drive voltages required by these devices, they are comparably easy to drive using a microcontroller. This presents the interesting possibility (if you don't get too carried away with trying to drive too many characters) of driving a VFD panel directly from the main system processor without the need and expense of a separate display module.

Of course, depending on the circumstances, you might decide to go ahead and design your own intelligent stand-alone VFD module, perhaps incorporating it onto the main system's PC board. If you just want to buy a display module, all you need is the know-how to interface to the panel and the required power source.

Table I--The *typical operating ranges of a vacuum fluorescent display make it suitable for many applications.*

cathode voltage	2.4–10 VAC (±10% rated)
cathode current	20–250 mA
grid/anode voltage	24–70 V p-p (±10% rated)
grid/anode current	2.5–30 mA
power dissipation	14–125 mW (per character)

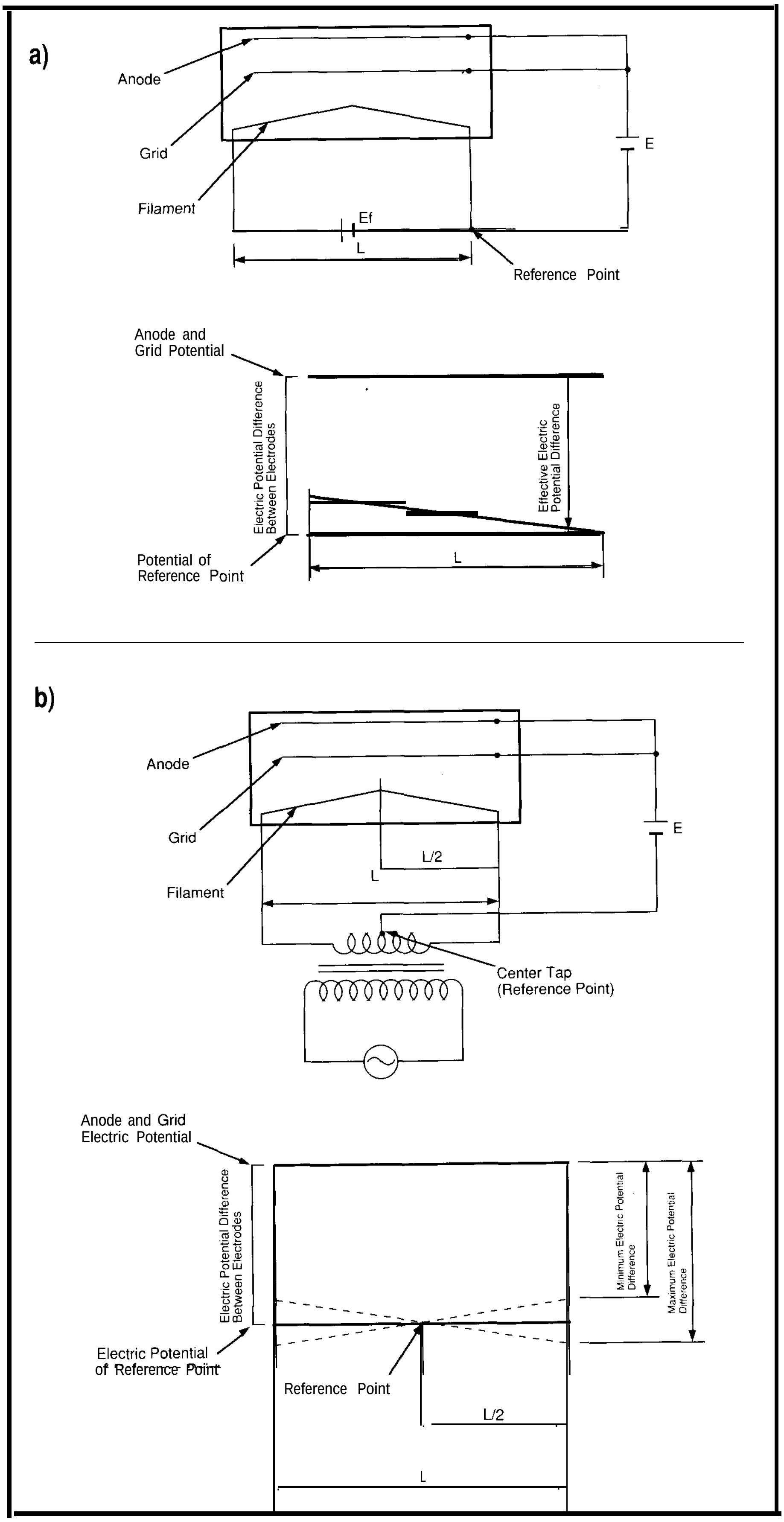

a)
Anode
Grid
Filament
E
Ef
L
Reference Point
Anode and Grid Potential
Electric Potential Difference Between Electrodes
Effective Electric Potential Difference
Potential of Reference Point
L
b)
Anode
Grid
Filament
E
L/2
L
Center Tap (Reference Point)
Anode and Grid Electric Potential
Electric Potential Difference Between Electrodes
Minimum Electric Potential Difference
Maximum Electric Potential Difference
Electric Potential of Reference Point
Reference Point
L/2
L

Figure I--The VFD's cathode may be driven with either DC or AC voltage. (a) *When using DC, voltage slant will* cause a brightness difference along *the anodes, making it less useful with large displays. (b) When* using AC, voltage slant is *still* a problem, but *a* center-tapped transformer evens out the *effect to* me *point of being negligible.*

VACUUM DISPLAY TUBES

VFDs utilize the principles of triode vacuum tubes. The VFD is composed of a hot cathode (filament), control grid, and a phosphor-treated anode. The application of a positive potential to the grid and anode accelerates cathode-emitted electrons through the electrical field. The anode will emit light when the electrons impact the fluorescent material on the anode with sufficient energy.

Typically, the light output contains all wavelengths in the visible spectrum, predominating in the blue-green area, with peak emissions in the 500-nm range. Because the emission spectrum is wide, you can obtain other colors by using optical filters or by changing the fluorescent material used in the VFD's construction.

A glass envelope provides the vacuum environment required for proper operation. The cathode is an oxide-coated tungsten filament that emits free electrons when heated (to about 600°C) by a current. The bias conditions placed on the grid and anode attain control of the display. For dynamic driving-that is, multiplexed modes of operation-the grid provides column (character) selection, while anodes manage individual segment control. Electrons from the cathode pass through it when a positive grid potential exists. If the anode is biased to a positive potential, it attracts electrons passing through the grid and is excited to luminesce. Table 1 summarizes the typical operating ranges for VFDs.

DRIVE REQUIREMENTS

Using DC or AC methods (sinusoidal or pulsed waveforms in the case of AC drive) can provide the cathode drive voltage. Usually this voltage must be within ±10% of the rated value. Exceeding this range reduces the operating life of the tube due to the evaporation of the oxide-coated material on the tungsten cathode from overheating. A low cathode supply means a poor display appearance because of the resulting lower temperature. Under normal conditions, the cathode's operating life is typically 50,000 to 70,000 hours. When an AC line supplies cathode power via a filament transformer, the operating frequency can range from 50 to 60 Hz. Using a DC-to-AC inverter, the upper limit is 1 MHz, a result of the effect of the cathode inductance.

Using a DC voltage for the cathode is possible; however, with this method a voltage drop (also called a *voltage slant*) develops between the positive and negative terminals, affecting the grid and anode bias. You will observe a brightness difference on the anodes traversing the varying potential along the cathode. For VFDs specified to operate with a DC cathode voltage, special steps are taken to minimize this effect, which is why the polarity is specified for DC cathode leads. This process becomes difficult for larger panels, so DC drive is usually used only for smaller displays.

The phenomenon resulting from this voltage slant exists in AC-driven cathodes as well. By using a center-tapped transformer for the cathode supply, the anode and grid voltage is referenced to the transformer's center tap, thereby reducing this effect to the point of being negligible. Figure 1 shows these effects as they pertain to DC and center-tapped AC cathode arrangements.

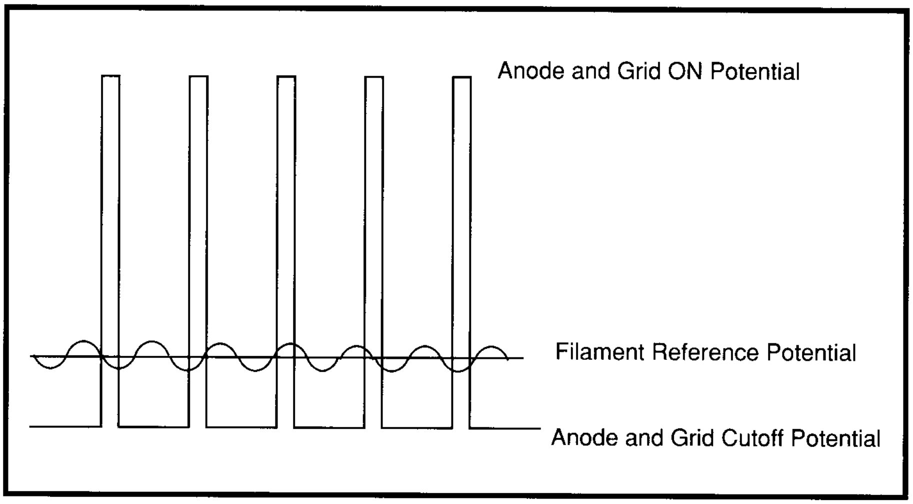
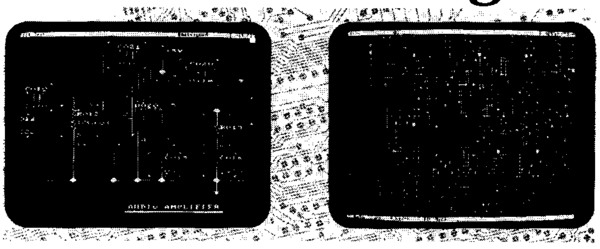
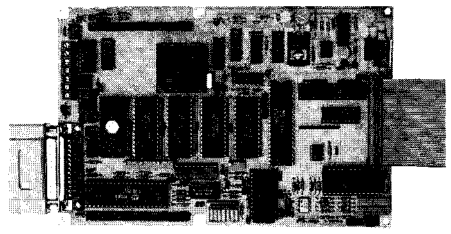

Anode and Grid ON Potential
Filament Reference Potential
Anode and Grid Cutoff Potential

Figure 2—*The anode and grid require a relatively high potential to begin conducting and, hence, luminescing. Lowering the potential to that of the filament isn't enough to stop conduction, however. A negative bias is required to extinguish the display.*

Assuming that the VFD is biased to luminesce, gradually decreasing the grid voltage eventually causes the display to extinguish. This point is called the *grid cutoff voltage.* The same situation exists when the anode voltage is reduced and is referred to as the *anode cutoff voltage.* In order to suppress luminescence completely, apply a negative voltage, referenced to the cathode potential, to the grid or anode, which prevents the movement of thermal electrons. Figure 2 illus-





trates the relation between the cathode, grid, and anode voltage levels.

Generally, the power required by a VFD is derived using one of two methods. In the first category, commercial AC voltage is transformed and rectified to generate the three voltages required by the VFD and controlling logic. These voltages include an AC center-tapped cathode supply, a negative grid/anode bias, and the regulated low-voltage logic power supply. In the second category, a DC voltage source, usually the logic power, is converted to provide an AC center-tapped cathode supply and a positive high-voltage grid/anode supply.

Though the AC-powered supply generates a negative grid/anode voltage and the corresponding voltage from the DC-sourced supply is positive, realize that the polarity is merely a matter of convenience. The polarity of the grid/anode voltage is meaningless, because the VFD itself is not ground referenced and the significant factor is the value relative to the reference point, which

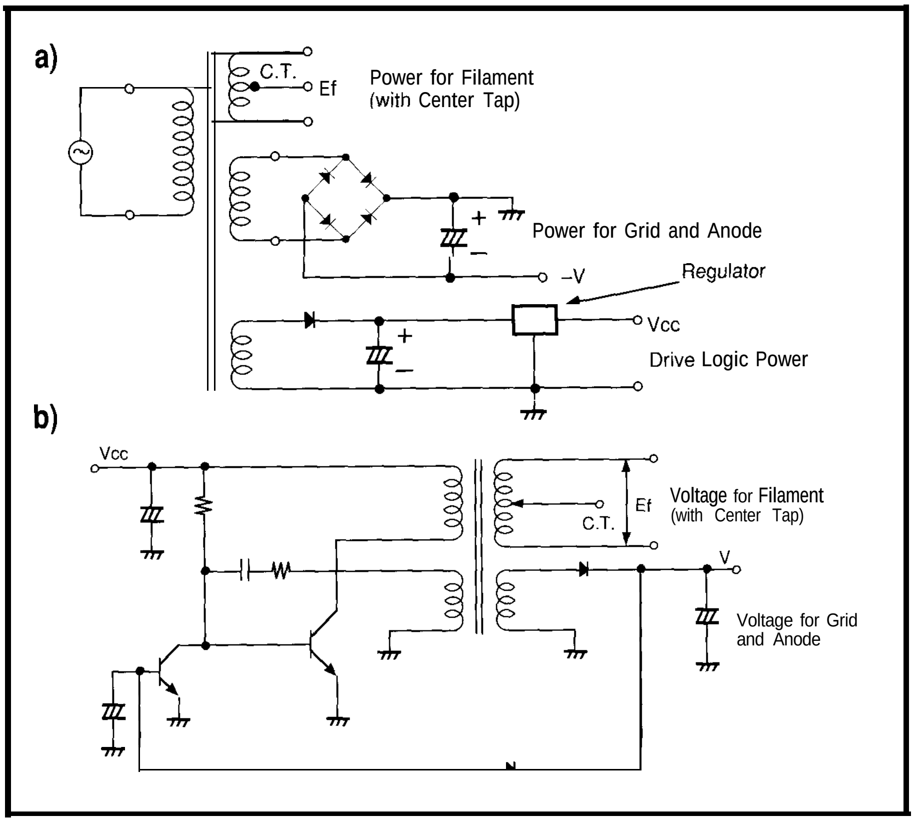
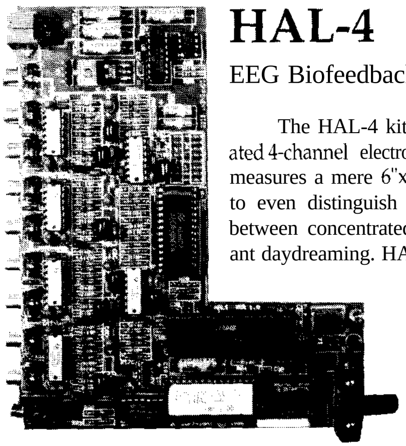


Figure 3-(a) *The power supply for an AC-powered display is usually derived from the 120-VAC power* line. A *single multitap* transformer can be used to run both the *display* and the 5-V drive logic. *(b)* When DC is used to power the display, it *must be* converted to AC and stepped up to the required levels.



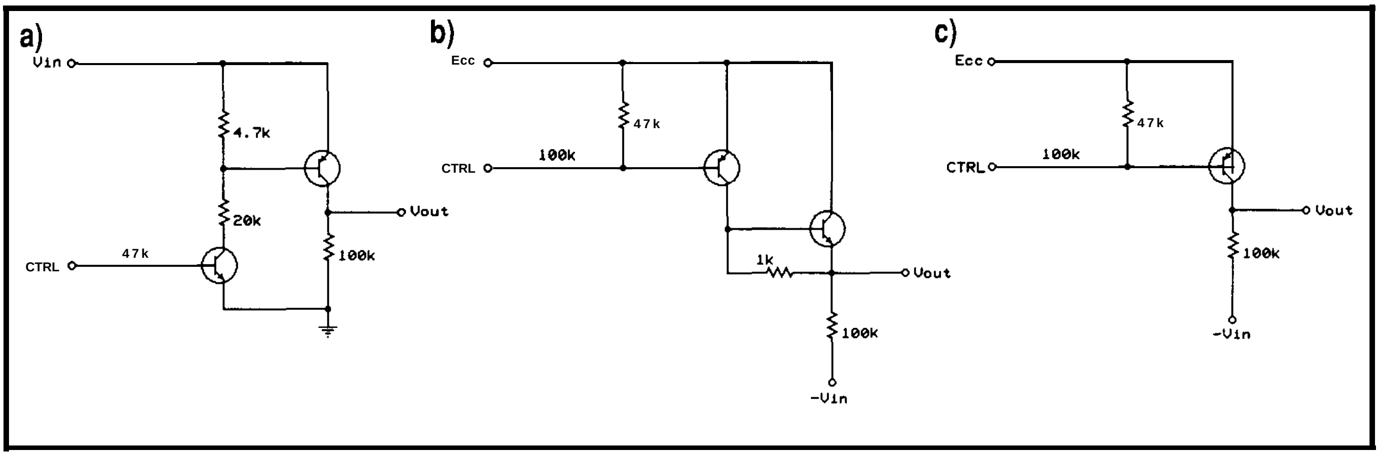


Figure 4—*Commercially* available integrated VFD driver circuits include (a) a ground referenced buffer, *(b)* a negative referenced buffer with increased current capability, *and (c) an ordinary negative referenced buffer.*

is the cathode. Figure 3 shows how to implement these two different power supplies.

STATIC AND DYNAMIC DRIVE

The minimal drive arrangement consists of an array of anode drivers used to select the segments to be illuminated by switching a positive voltage. This arrangement is the so-called *static drive mode* in which the grid is tied to the same source voltage switched via the anode drivers. This method is effective only for small displays because the number of electrode terminals increases rapidly with additional characters.

Dynamic driving uses the familiar multiplexing scheme in which a set of segment connections is common throughout the display and individual character positions are selected by energizing the grid electrodes in a row/column-type matrix. In this design, the grid is sequentially scanned with the segment data registered in time so the display of each digit is intermittently repeated. This operation repeats more quickly then you can sense visually, so the display appears continuously illuminated to the eye.

A wide variety of integrated driver circuits is available for switching the grid and anode voltages. Because the potential between the cathode and the grid and anodes is what is important, the drive scheme can be ground based, or a negative potential can be used. The result is the same in any case, and the choice is based on the overall power supply configuration and ease of implementation. Figure 4 shows the basic functions performed by commercial-integrated VFD driver circuits using both positive and negative supply voltages.

THENEEDFORSPEED

Several different conditions may contribute to the appearance of flicker when driving a VFD dynamically. Because the human eye begins to notice flicker at about 40 Hz, a display with a refresh rate less than that will clearly exhibit this effect. This situation would be considered a design problem because the limitation is known.

Assuming a 60-Hz cathode frequency (which is what you'd get using a filament transformer), a second type of flicker with a rolling appearance arises at refresh rates between 40 Hz and 90 Hz. This condition, called *beating,* is a result of the cathode and refresh frequencies being too close together. Here's what happens. If a character is on only during the time the cathode voltage is negative, it may appear brighter than the character next

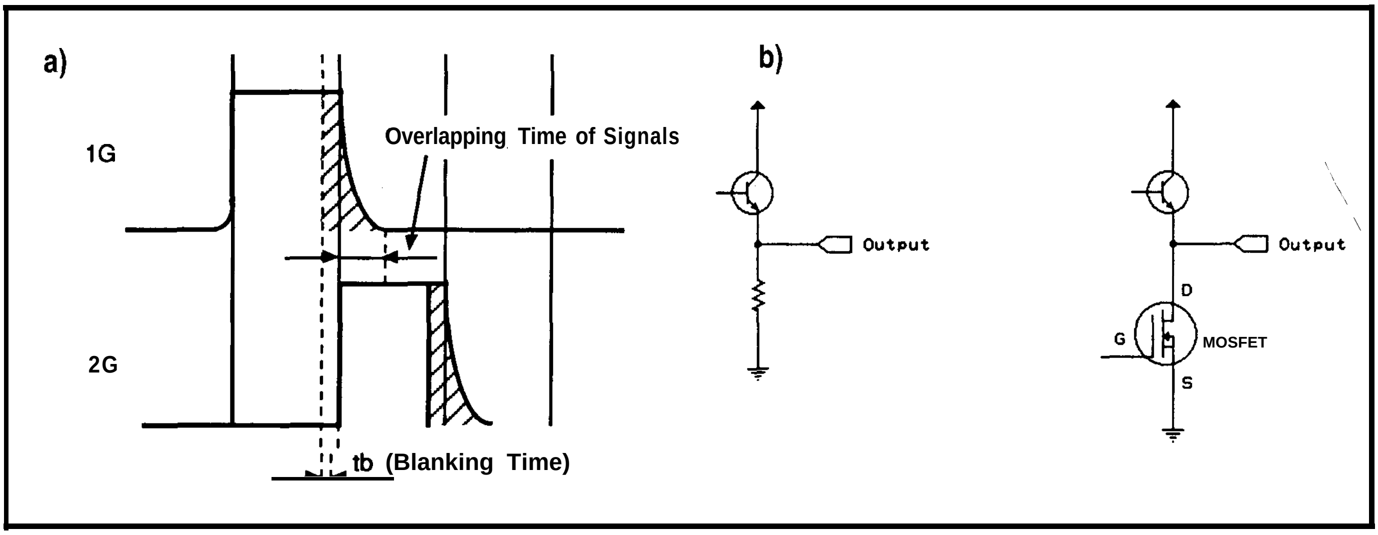


Figure 5—*(a) Rise and fall time characteristics of the voltage waveform result in crosstalk. (b) Using conventional drivers with pull-down resistors, dead time can range from 20 to 30 µs, but no less than 10 µs. To reduce this time, an active pull-down element is necessary.*

to it, which may be on only during the positive cycle of the cathode voltage, because of a resulting net difference in grid/anode to cathode potential.

You can avoid this condition if you keep the difference between the cathode frequency and the scan frequency to more the 30 Hz. If the controller can hack it, the easiest way to correct this problem is to increase the scan rate. If doing so is not possible, then you're forced to increase the cathode frequency, which usually requires the use of some sort of DC-to-AC converter for the cathode power.

When selecting the refresh rate, consider that the display may not necessarily be viewed by eyes fixed in the same position or that the display itself may not be fixed. For example, you may need a faster refresh rate in cases where vibration is present. Setting the refresh rate to at least 100 Hz for displays installed in a fixed base is probably a good idea. For displays operating where vibration is apt to occur, such as in vehicles, you may need to increase the refresh rate to 400 Hz. Of course, there are limits to the refresh rates attainable using microcontrollers; generally, you'll want to perform the refresh as fast possible.

VFDs are pretty forgiving from a timing standpoint; panel size primarily determines the actual timing. As I mentioned, the minimum refresh rate with undetectable flicker is about 60 Hz (and this level is pushing it). This parameter defines the refresh time (T_r) permitted to update the entire display frame. The number of grid control lines determines the amount of time allotted for each grid, or character, operation (T_c). In order to maintain acceptable display intensity, each character position should operate at a minimum duty cycle of 1:150 ($T_p = T_r$ × duty cycle).

Not considering any intercharacter blanking for the moment, a display could contain as many as 150 grid control lines (characters) if you accept these constraints. However, allot some dead time for intercharacter blanking (T_b) to prevent ghosting. The intercharacter blanking time equals the character time minus the character pulse width ($T_b = T_c - T_p$).

The crosstalk resulting from the rise and fall time characteristics of the voltage waveform causes ghosting. As the grids are scanned sequentially, a period of time will occur when two adjacent grids will both be on at the same time, unless you impose a blanking interval to allow the grid voltage to decay sufficiently before the next one is energized. Using conventional drivers with pull-down resistors, this dead time must be on the order of 20 us to 30 us. However, you can shorten this time by using drivers with an active pull-down element. In any event, the need for blanking limits the maximum number of display characters to fewer than 150. A good practical upper limit would be 128, a fitting binary number. Figure 5 shows how crosstalk occurs.

As an example, suppose I want to drive a 40-character VFD at a refresh rate of 200 Hz. (Remember, the higher the scan rate the more stable the VFD appearance.) The time allotted to refresh the entire display is $T_r = 5$ ms. Therefore, the amount of time for each character is $T_c = 5$ ms / 40 = 125 µs. Assuming a 1:50 duty cycle, 25 us is left for the blanking time, where $T_p = 5$ ms / 50 = 100 µs and $T_b = T_c - T_p$. These figures indicate that this rate of refresh falls within the capability of a microcontroller, although a slower refresh rate would be more comfortable and would generally work fine. The lower refresh rate would also free up the controller, giving it time to do something besides scan the VFD. If you had to scan more grid positions, setting up the circuitry to drive two sets of grids in tandem while updating the anodes for the respective two characters in one shot is a good idea.

If you work the numbers for a larger display, the need for drivers with an active pull-down component becomes obvious. Say you want to scan a display with 128 grids. Dropping back to a 100-Hz scan rate, T_r is 10 ms. With 128 grids, T_c comes to 10 ms / 128 = 78 µs. Reducing the duty cycle to 1:150, T_p is now 10 ms / 150 = 66 us. The time remaining for T_b is 78 us – 66 µs = 12 us, which is beyond the capability of drivers relying on passive pull-down resistors. This time also puts you on the hairy edge of what you can expect from a cheap microcontroller, so you'll most likely need additional hardware to pull it off.

DIMMING

One of the advantages of using a VFD is its bright output capability, which allows easy visibility in various lighting conditions. However, the VFD's output can be excessive in subdued ambient lighting conditions and downright dangerous in certain environments, especially if used in automotive applications.

Although varying the VFD's brightness by varying the grid or anode voltage is certainly possible, doing so is not the optimal way of handling the situation. Using this approach has constraints; you may not limit the voltage beyond 10% of the rated value. Going below this range could cause problems in the uniformity of the display output. Also, you would need a way to control the voltage, which would require additional circuitry.

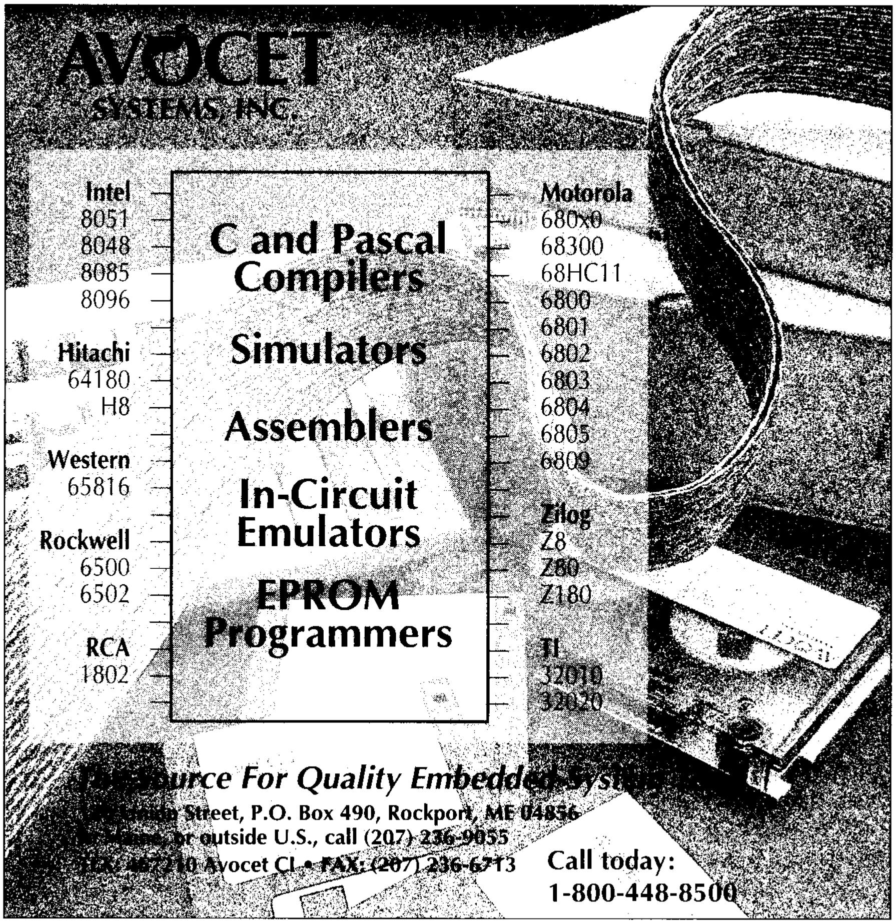
Controlling the grid duty cycle is a better way to handle VFD dimming. This method works well because the driving voltage doesn't need changing. If you're using a microcontroller to perform the display scanning, the intercharacter blank time must be managed anyway. So by further controlling the grids' pulse width, you can dim the display using software without too much additional trouble.

I'll wrap up my discussion of VFDs next time by presenting a project that actually works a display tube. As usual, I'll look at the practical considerations and tradeoffs of such an exercise and maybe even come up with the proverbial neat hack. ❑

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

IRS

422 Very Useful
423 Moderately Useful
424 Not Useful



PATENT TALK

by Russ Reiss

This month addresses a varied collection of patents you may find interesting. They range from the latest PAL design tool by Minc Inc. to a "low-tech" patent by IBM. Also, I include a software patent from Microsoft and a novel combination of microprocessor and IR link in the form of an “electronic book.”

The Minc patent came to my attention when they referred to it in their data sheet, which recently came across my desk. It represents the next step in PAL design tools. As you will see by perusing Abstract 1, Minc not only translates the Boolean information into PAL programming format, but goes one step further and determines the optimum type and arrangement of PALs for implementing a given system. The user defines what he or she means by “optimum” through creating weighting factors for the various design criteria. This approach should save you considerable time and effort and result in more dense system designs wherever you use a significant quantity of PAL-type devices.

At the other extreme, I was amused by IBM’s recent (August 1992) patent in which they (finally] addressed one of the major shortcomings of the original PC design: insufficient cooling of boards and components. While I appreciate the use of dual intake and exhaust fans in a PC (as Abstract 2 presents), I can’t help wondering which small “industrial computer” manufacturer will be tempted to challenge the giant on this simple improvement. Obviously, designing this change was a major engineering effort, because the patent is issued to no fewer than *five* inventors!

A search of recent patents from Microsoft (thirteen of them since the beginning of 1990, but eight of these in the first half of 1992 alone) reveals their movement into the area of computer applications and not just operating systems and languages. Abstract 3 represents a patent using a collection of known techniques combined into an apparently novel text compression and decompression method. Of particular interest is this approach’s ability to retrieve data a line at a time with a single decompression pass. If the compression ratios achieved in this patent are significant, I can only surmise what interesting applications they have in mind for future products.

Finally, a search of microprocessor applications involving infrared (IR) technology turned up what I thought was an interesting application-an electronic book! The inventor (see Abstract 4) linked a PC with mass storage to remote display stations via an IR link. I would imagine users in a classroom, library, or business environment would “read the book” on the screen while the data passes over the IR link. By providing local memory in the display station, the user may continue to read the book (or sections of it) while out of range of the host PC.

This possibility conjured up in my mind a scenario of the modern student, notebook computer (with IR link) in hand, accessing all types of reading material throughout the day, to address in greater depth back at the dorm in the

1

Patent Number	5,140,526
Issue Date	19920818
Inventor(s)	McDermith, William 0.; Banki, Mehrdad; Bush, Kevin M.
State/Country	c o
Assignee	Minc Incorporated
Title	Partitioning of Boolean logic equations into physical logic devices
Abstract	An automated system for partitioning a set of Boolean logic equations onto one or more devices selected from a plurality of commercially available devices. The system utilizes a processor having a memory containing information on the different architectural types of devices, physical device information on individual devices and user generated design constraints, weighting factors and partitioning directives. Based upon this stored information, the system of the present invention selects all acceptable architectural types of devices wherein at least one of the Boolean logic equations can be placed thereon. For all physical devices associated with the acceptable architectural types only those devices which fall within the selected user constraints are selected. The system then evaluates the weighting factors to order the devices in order of cost value and then fits the equations according to the partitioning directives to the devices. During the fitting process, an optimum device solution is attained having a least cost value for which the system produces an output map suitable for the user of the system to configure the selected devices to implement the set of equations.

PATENT TALK

2

Patent Number	5,136,465
Issue Date	1992 08 04
Inventor(s)	Benck, Jeffrey W.; Mansuria, Mohanlal S.; Miller, Michael S.; Musa, Richard D.; Trumbo, Brian A.
State/Country	FL
Assignee	International Business Machines Corp.
Title	Personal computer with tandem air flow dual fans and baffle directed air cooling
Abstract	This invention relates to personal computers in which provision is made for effective cooling of components capable of generating significant heat during operation, such as certain high-performance microprocessors. The personal computer has an enclosure for enclosing operating components, a printed circuit board mounted within the enclosure for supporting and interconnecting operating components of the personal computer, heat-generating components mounted on the printed circuit board for performing operating functions for the personal computer, a fan for inducing air to flow into the enclosure, a fan for expelling air from the enclosure, and an air-flow-directing baffle mounted within the enclosure in the path of air flow from one fan toward the other fan and adjacent to the heat-generating components for directing the flow of air through the enclosure to pass over and cool the heat-generating components.

evening. While in class, he or she could download lecture reference material and current assignments automatically, then add background material during a visit to the library. After picking up a copy of the campus daily newspaper at the student union, the student would return home to read an electronic book, with all the benefits of text searching, cut-and-paste editing, and hard-copy printing at his or her fingertips.

Remember, this column is for *you.* Please give me feedback on topics of interest to you.

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

IRS

425 Very Useful 426 Moderately Useful 427 Not Useful

3

Patent Number	5,109,433
Issue Date	1992 04 28
Inventor(s)	Notenboom, Leo A.
State/Country	WA
Assignee	Microsoft Corporation
Title	Compressing and decompressing text files
Abstract	A method of compressing a text file in digital form is disclosed. A full text file having characters formed into phrases is provided by an author. The characters are digitally represented by bytes. A first pass compression is sequentially followed by a second pass compression of the text which has previously been compressed. A third or fourth level compression is serially performed on the previously compressed text. For example, in a first pass, the text is run-length compressed. In a second pass, the compressed text is further compressed with key phrase compression. In a third pass, the compressed text is further compressed with Huffman compression. The compressed text is stored in a text file having a Huffman decode tree, a key phrase table, and a topic index. The data is decompressed in a single pass and provided one line at a time as an output. Sequential compressing of the text minimizes the storage space required for the file. Decompressing of the text is performed in a single pass. As a complete line is decompressed, it is output rapidly, providing full text to a user.

PATENT TALK

4

Patent Number 4,855,725
Issue Date 1989 08 08

Inventor(s) Fernandez, Emilio A.
State/Country VA

Title Microprocessor based simulated book

Abstract A user interactive mass storage data access system includes a personal computer (10) and a simulated book (30). A mass storage device, such as a compact disk (CD) read only memory (ROM) (22), is connected to the personal computer, and the computer and the simulated book are connected by an infrared (IR) data communications link including IR transceivers (26, 48). The simulated book includes a display screen (34) and a microprocessor (43) with memory (44, 46). The microprocessor is programmed for storing data received and decoded by its IR transceiver (48) in memory (46) and responsive to user input for displaying a page of data on the display screen. In addition, the microprocessor is programmed to cause its IR transceiver (48 I) to transmit to the IR transceiver (26) connected to the personal computer (10) a data request command, and the personal computer is in turn programmed to transmit data from the CD ROM (22) to the simulated book (30). Data can be loaded in the simulated book and accessed at a later time when out of the proximity of the personal computer.

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
Vernon, Connecticut

If you look carefully at this month's messages, you'll notice that the message numbers have suddenly shrunk. We finally bumped info the 16-bit limit on message numbers (65,535), so had to renumber the whole message base. The only drawback is if will be difficult for you to locate threads from past installments of ConnecTime should you want to add your own rep/y on the BBS. Such are the hazards of a continuously growing BBS.

I'll start this month off with a discussion of how to add a watchdog to an IBM PC. While the thread talks about fhe PC specifically, the techniques can be used to add a watchdog to just about any computer. Next, we'll fake a look at some possibilities for implementing a radio direction finder. The third thread is a conversation about lock-in amplifiers and phase-sensifive defectors. Finally, we venture info high-frequency programmable oscillator design.

Msg#: *6488*
From: PHIL COMBS To: ALL USERS

I am interested in finding or making a (reasonably priced) watchdog board for an IBM PC. I wish to keep an unattended PC from locking up and such a board would be necessary. I would like to be able to control it with program code I could link into my software.

I have looked in Maxim's catalogs and noticed they have microcontroller "supervisory" chips that include watchdog reset timers. I might be able to figure out how to interface one to a PC, but figuring out how to interface the bugger in software is beyond me. Does anyone have any experience in this area?

Msg#: 6517
From: ED NISLEY To: PHIL COMBS

Having a watchdog in an "unattended" PC is a good idea; the trick is using it correctly so it can detect hangs.

Most PC system boards now come with a "reset" input, so you'd just wire the watchdog's output to that input (with the right polarity) and you're all set. One of the Maxim chips even has a reset push-button debounce circuit, so you could still use the front-panel push button!

You need to guarantee that your code will ***always*** toggle the watchdog's input every 50 ms or so. The exact interval depends on which watchdog chip you're using, but it's a good idea to toggle it a lot more often that required, just to make sure you don't get reset when you underestimate how long a routine will take.

A trick I've used with 803 1 systems is to set up a 16-bit variable in RAM, then have a timer interrupt shift the bits out to a heartbeat LED and the watchdog. When each bit goes out, the code produces a little glitch so the watchdog sees a transition even between O-O and l-l bits. When all 16 bits are done, the interrupt handler sets a flag that must be cleared by the time it sends the bits a second time. If the flag is still set 16 bits later, the timer interrupt locks up and the watchdog resets the system.

The main-loop code simply clears the flag every now and again; if the timer is ticking every 100 ms, you have 1.6 seconds for the main loop. That's easy enough to do and will still reset the system in a few seconds if your main loop crashes.

Don't just put the watchdog toggle in the interrupt routine because the main loop may crash while the timer keeps on ticking.. .the watchdog will be perfectly happy, but you'll be ever so sad!

Make any sense?

Msg#: 6553
From: PHIL COMBS To: ED NISLEY

Yes, I believe what you said makes sense. My main question now is, what are the fine points in interfacing a Maxim watchdog to the PC bus, and could such a circuit be attached somewhere else besides the PC bus (e.g., under the keyboard ROM or a BIOS ROM, saving a PC slot)?

Msg#: 6636
From: ED NISLEY To: PHIL COMBS

Turns out that it doesn't matter where you hitch the watchdog as long as your program can control the outputs to it.. .or as long as the outputs will stop when the system locks up.

You need only one bit, so you could wire it to, say, the speaker output bit and toggle that bit once in a while. It wouldn't be audible, because you'd flip the bit back and forth before the speaker could respond!

But using a nondedicated bit can have problems. Suppose the code locked up in the "sound a tone" subroutine: your PC would howl until you came in to flip the Big Red Switch. That's why you need to think about how to produce that output.. .it's not simple!

CONNECTIME

Msg#: 7386
From: ALAN COOK To: PHIL COMBS

Sorry for butting in guys, but, for what it's worth, my two pennies' worth.

The only reliable way that I have found of using a PC watchdog-type of circuit is to map it into the I/O area. This simplifies the resetting of the watchdog circuit since it is only necessary to address it using, for example, `outportb(0x230,any char)`. The reset signal for the watchdog can then simply be derived from a chip select signal representing that port address.

Incidentally, I used a PIC micro, rather than a dedicated watchdog device, since it allows me considerable flexibility in configuring the watchdog period, and, having its own watchdog timer, can recover itself if the need arises. It also has other I/O capability that allowed me to add additional functions to the PCB.

While commercial direction finders are readily available, the do-it-yourself spirit is alive and well on the BBS. Here, we look at some design possibilities.

Msg#: 6366
From: WAYNE FUGITT To: ALL USERS

Has anyone any experience with circuits relating to short- or medium-range direction finders?

A friend had asked me to build a direction finder for him. I realize something can possibly be purchased off the shelf. That would be no fun for an experimenter. We should be able to build a transmitter and receiver for maybe $50 to $60 parts.

The only question I have right now is what frequency range should I work with! The final output power may be less than 1 watt. But in case we needed more power, I would not want to be handicapped by using the wrong frequency.

Msg#: 6372
From: RUSS **REISS** To: WAYNE **FUGITT**

There are many approaches to direction finding, and a lot depends on what you define as "short or medium range." I presume you are talking just on the order of a mile or so. Certainly, for that range, you will want to use VHF/UHF frequencies.

If it is suitable for you to rotate an antenna, you can determine the general direction of a remote transmitter by the peak or the null of a directional antenna as it is rotated. This need not be a complex setup either. The ham (amateur radio) literature shows many designs. The idea is that the antenna should present a very sharp null (typically] which can be detected. This shows you the direction of the transmitter.

On the other hand, if operation must be automatic, the system used by many amateur repeater operators to pinpoint illegal interferers might be appropriate. One popular and inexpensive scheme uses a pair of antennas switched at an audio rate. The resulting output from the speaker is an audio signal, the phase of which determines the direction of the station. This is a patented system as I recall, but don't remember by whom. You may build one for your own use even though it is patented.. .just don't sell them!

If you feel you need over 1 W (actually over 100 mW!), you'd better plan on licensing that transmitter! There is NO unlicensed VHF/UHF transmitter operation over 1 W. How about getting one of the new "no code" ham technician licenses? That would get you (and your friend) operating with all the privileges you need for what you want to do (plus a LOT more).

Msg#: 7206
From: WAYNE FUGITT To: RUSS **REISS**

Thanks for the detailed reply re the radio direction finder. You pointed out a few things that I was not up-to-date on.

I suppose with some fine tuning, I could get by with the l-watt limit. The terrain where this would be used consist of hill and heavy timber. These obstacles would likely reduce the range somewhat.

Msg#: 6533
From: WILLIAM VONNOVAK To: WAYNE **FUGITT**

Aircraft radio navigation systems sometimes use ADFs (automatic direction finders) that operate around 1 MHz. There should be loads of schematics around for them—they've been around since World War 2 (they've mostly been replaced by the VOR receiver). There are also many handy NDBs [nondirectional beacons) around on which to test the receiver, and you can use an AM station as a beacon in a pinch.

It seems you can get single chips to do just about any function these days. The search for a single chip to implement a lock-in amplifier or phase defector isn't as easy as you might think, though.

Msg#: 6463
From: FRANK ROE To: ALL USERS

Does anybody know of an IC that has all or some of the functionality of a lock-in amplifier or phase-sensitive detector? I tried scanning past messages on the bulletin board and had no luck.

CONNECTIME

Such a chip would have two inputs: one would be a noisy analog signal, the other a reference frequency. There would be one primary output: a signal (AC or DC) with amplitude proportional to the component of the noisy input and that has the same frequency as the reference. Another possible output would be the phase shift between the reference and the component.

Msg#: 6565
From: DALE SINCLAIR To: FRANK ROE

I just had a look through the schematics of my Stanford Research Systems lock-in and no, there ain't no easy way. You've basically got a phase-sensitive detector fed by a reference sine wave and the signal of interest followed by filters. The reference path involves a phase-locked loop, phase shifter, and sine wave converter to get a useful reference, and the signal path has a lot of filters and low-noise preamps before it gets to the mixer. Most of the chips are LF411,LM311, and so forth, but I don't see a big "lock-in" chip at all. My guess is that those devices popular for industrial use just don't cut it for scientific applications.

Msg#: 6712
From: RUSS **REISS** To: FRANK ROE

I'm not familiar with such a chip either, but out of curiosity, what range of frequencies are you interested in?

Msg#: 6998
From: FRANK ROE To: RUSS **REISS**

The frequencies are not highly demanding: 100 Hz to 1000 Hz. Anything else is icing on the cake.

Msg#: 6733
From: PELLERVO KASKINEN To: FRANK ROE

No need to jump so quickly to completely negative conclusions! Depending on the frequency range you are looking for, there are several avenues to take. At first, let me sort of define a couple of terms. One item is the basic phase-sensitive detector, the other is a lock-in amplifier.

A classical example of a phase detector is the XOR gate. Depending on the implementation, you may get in-phase or quadrature operation. A commercial chip, RCA CD 4046, contains the detector and, as a bonus, even a voltage-controlled oscillator for a phase-locked loop. What you need to do is provide a "loop filter" that has to offer suitable speed of response and bandwidth to assure a lock-in and at the same time minimal hunting. A principal feature of a phase detector is that it uses fully clipped input signals, both the "noisy" one and the reference (local oscillator or injection are common terms for the next part of the story). Therefore, you cannot use the output directly for a measurement of the input signal amplitude; you need another device, such as an analog gate (4066, DG412, or somesuch) that feeds synchronous samples of the input signal into a low-pass filter. The bandwidth around the local oscillator frequency is twice the low-pass knee frequency.

To preserve the signal amplitude information in the first place, you can use an analog multiplier chip, such an AD734 (up to 10 MHz) or AD534 (up to 200300 kHz}. In that case, you do not clip the signal or the injection but try to keep the sinusoidal quality. If you have a perfect match between two signals, the result of the multiplication is a double frequency (want to check the trigonometry?) that rides on a DC level. This DC level reflects the instantaneous phase relationship-highest for phase match, zero at a quadrature relationship, and most negative at 180° phase shift. The important consequences are that given a stable injection amplitude, the DC level is also proportional to the signal amplitude. Adding a low-pass filter of known gain (like unity) allows you to extract the in-phase component from your noisy signal in a "calibrated" fashion.

Now, just like the phase detector characteristics depend on the loop filter, the low-pass filter after the multiplier determines the bandwidth of your tuned (i.e., frequency selective) measuring system. If you make the filter very narrow, then do not expect it to handle fast slewing frequencies from either the input or the local oscillator properly.

To widen the perspective a little more, these concepts are regularly used in amateur radio circles and were at the top of list for interest at the time of popularizing the SSB (single side band] modulation. Look for terms such as "product detector" or "balanced modulator." Actually, you might want a double balanced modulator to minimize both the local oscillator and the input signal feedthrough. But that is something I let others handle if necessary.

Now, a word about the commercial lock-in amplifiers. They are more complicated than my introduction appears to indicate for several reasons. The main reason is that we often want the whole amplitude, rather than just the in-phase component. That requires a second detector with a 90" off set phase reference signal and then a square root of the two signals squared (i.e., a vector sum), or two meters to display the in-phase and the quadrature result separately. So, natural complications, but you can pick up how far you care to pursue it. I have performed some measurements with a makeshift arrangement that might be of interest here.

Msg#: 6999
From: FRANK ROE To: PELLERVO KASKINEN

Thanks for the detailed reply to my question. It appears to be very helpful. I have been using a commercial LIA, but it has more features than I need right now.

CONNECTIME

Speaking of looking for single-chip solutions, it would appear that trying to find a single chip to generate a very specific kind of clock is just as difficult as in the previous thread.

Msg#: 7960
From: **LOAI** NAJJAR To: ALL USERS

I would appreciate help on a design problem. I need to design a programmable clock from 5 MHz to 64 MHz in 0.5-MHz increments. The duty cycle must be programmable from 25% to 75% in 5% increments. Do any of you know of a chip that meets these requirements? Currently I have a design using delay lines which is programmable, but because of inherent delay, the maximum frequency is 40 MHz. I could put a PLL stage on it and go up to 64 MHz, but then the duty cycle will be 50%.

Msg#: 8021
From: JAMES MEYER To: **LOAI** NAJJAR

Those are some *very* tight specs you have there. The difference between 45% and 50% duty cycle at 64 MHz is only 0.78 nanoseconds.

If I had to do the job, I'd make a 1280-MHz oscillator and feed it into a programmable counter. Then I'd count down from a preset number representing the time I would want the clock to be high and feed the zero count into a flip-flop. At the time I got a zero count, I'd also preload the counter with the time that I'd want the clock to be low. This sequence would be repeated over and over. The output of the flip-flop would be the clock signal that I wanted.

All the logic would have to be GaAs-FET or maybe ECL, and you would need stripline PC board layouts to get it to work. But it would be 100% accurate and require no "tuning."

Msg#: 8035
From: DAVE TWEED To: JAMES MEYER

That's fine, except you missed a key specification: The basic output frequency needs to be settable to 0.5 MHz. Your circuit gives us 64-MHz fine at 5% to 95 % duty cycle, but the next lower frequency it can produce is 60.95 MHz, and *none' of the duty cycle settings are multiples of 5%.

If you want to go with all-synchronous digital logic, you'll need to use a technique called DDS, or Direct Digital Synthesis. It's common in the newest communication gear, especially "spread spectrum." You can buy chips off-the-shelf from Qualcomm and others that do this (but possibly not all the way up to 64 MHz).

The basic idea is that you take a number that represents the output frequency and repeatedly add it to an accumulator. The high-order bits of the accumulator represent the phase of your output signal. The period of the signal is the number of clocks between accumulator overflows. The frequency resolution is limited only by the number of bits you use to represent the frequency. You can then feed the phase value to a ROM and then to a D/A converter to generate any arbitrary waveform (sine is common), or you can use the phase value directly to generate a ramp waveform.

To get your variable-duty-cycle rectangular waveform, you compare the ramp value with a fixed level so the output is low when the ramp is less than the reference level and high when it's higher. This last part can be done in the digital or the analog domain, depending on the requirements. If you do it in the digital domain, the output can change state only on the clock transitions of the accumulator (since the output is constant between clocks). This causes jitter in the placement of the edges of the output waveform, resulting in a large amount of phase noise.

If you do it in the analog domain by using a video DAC, a filter, and a high-speed comparator (with the reference level set by a separate low-speed DAC), you get much finer resolution at the expense of a more complicated design. On the other hand, this technique can usually allow you to use a slower-speed clock to the accumulator.

Msg#: 8100
From: JAMES MEYER To: DAVE TWEED

Picky, picky, picky.. OK, you're right. But just look at the resolution you'd get around a 1 -MHz output. Your way will work, but I think it's overkill. There *must * be a way that's somewhere in between our different approaches that will work.

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

[redacted]

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

To order Software on Disk, send check or money order to: The Computer Applications Journal, Software On Disk, P.O. Box 772, Vernon, CT 06066, or use your VISA or Mastercard and call (203) 875-2199. Be sure to specify the issue number of each disk you order. Please add $3 for shipping outside the U.S.

[redacted]

428 Very Useful 429 Moderately Useful 430 Not Useful

STEVE'S OWN INK

PC Trials

Ohere's an ad currently running on TV showing a couple guys spending days trying to get a PC up and running. The frustrating scenario includes quoting the cryptic operating manual to each other, obviously hoping that mere repetition will result in understanding. The ad ends with another business partner saying that if they take any longer to get the computer that's supposed to improve efficiency working, there won't be any business to improve.

Unfortunately, this ad about PCs is all too true. In fact, I feel just like those guys in the ad.

A few of weeks ago I decided to upgrade a couple systems for some projects. Since the 386SX/16 I'd been using for the last few years was adequate for one of those projects, I took the opportunity to buy a new 486DX/50 system. Being adventurous, I even decided to add a CD-ROM and other peripherals myself.

The system came preloaded with Windows and some other programs. I attached the SVGA monitor, keyboard, and mouse, and flipped the power switch. Voilà! The Windows display appeared. The scuttlebutt about PCs was all wrong. This one obviously worked.

With my confidence still intact, I next decided to add the SCSI CD-ROM drive. I inserted the SCSI expansion port card, connected the drive, and ran the install program. Two hours later, after diagnosing a conflicting IRQ designation, I got the system to acknowledge that a CD-ROM existed. Various versions of CONFIG.SYS and AUTOEXEC.BAT littered the hard drive directory.

With things looking up again, I inserted a CD-ROM, Street Atlas USA, and went to the CD-ROM drive from DOS to start it. Nope, this one only works under Windows. OK, boot Windows, mouse over to the file manager, run the installer for Street Atlas, and click on the resulting icon? Nope, crash! "Not enough memory" was displayed.

Of course, nothing in the Atlas or the CD-ROM drive directions say anything about memory. Wasn't 8 meg of RAM enough?

Three hours later, someone came up with an obscure note in the installation directions of a completely different CD-ROM drive that said, "some software may require as much as 615K bytes of available RAM" (640K bytes is all that's available). CHKDSK came back saying I had only 537K available. So much for preinstalled systems. Back to CONFIG.SYS.

After a few more exasperating hours, I called Ed Nisley and he said I needed QEMM386. After a trip to SAM's Warehouse, two more hours, and $80, I had QEMM. Believe me, at that point you are running on pure faith.

QEMM386 comes back periodically and says stuff like, "Shall I trash your old code and replace it with something else?" After doing this a few times, CONFIG.SYS is about 10 times longer and has command statements that only the original authors of DOS could understand. Finally, running the OPTIMIZE program adds another few lines of cryptic dribble that's supposedly worth $80.

CHKDSK now came up with 588K, but the Atlas still wouldn't run. As panic set in, I called Ed again. Fortunately, he makes house calls.

Ed opened up his portable next to my system, compared the two CONFIG.SYS files, and proceeded to change mine to match his. When quizzed as to the relevance of the entries, Ed simply replied that when something works on a PC, it is cause célèbre enough and questioning why is irrelevant. When he finished, CHKDSK reported 619K bytes available!

The broad smile evaporated instantly when Atlas came up with "Not enough memory!" again. What gives? Ed waved good bye.

Two hours short of three days, while still staring at the Street Atlas USA icon that I couldn't get to execute properly, I decided to run the install program again. At the conclusion, it painted the same icon. I clicked on it.

To my amazement, the program started and maps appeared. A quick mouse move and it scrolled to follow. It worked! But, what had I done other than spend three days? Or, was that just the rule that we all had to live with? I called Ed and he perceptively acknowledged, "Yup, three days."

Now, I'm sitting here with a color printer, scanner, and sound board wondering whether I should tempt fate or leave well enough alone. Hey, Ed, what'cha doing for the next few weeks?

Steve