CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

REAL-TIME PROGRAMMING

SPECIAL SECTION:
Embedded Sensors & Storage

Multitasking Kernel in C

Data Logger Design

Programmable Logic Controllers

June/July, 1992 — Issue #27

$3.95 U.S.
$4.95 Canada

EDITOR'S INK

Computing in Real Time

Most people will agree that when developing code for an embedded controller, there is usually some degree of "real-time programming" involved. It's that ever so tenuous balance between code complexity, code size, execution speed, and response time. Programs running on a desk-top machine can handle unrelated tasks in any order and can take their time doing so. Speed comes into play only when testing the user's patience.

However, when the tables are turned and the microprocessor must respond in a timely manner to any number of asynchronous external stimuli, code complexity increases rapidly unless the programmer uses different techniques when wriiing the code. Enter multitasking.

Multitasking kernels generally come in two flavors: preemptive and nonpreemptive. In our first article, past design contest winner Mike Podanoffsky shows how a simple preemptive multitasking kernel can be written in C. In a nonpreemptive system (like Microsoft Windows), tasks can't be interrupted, so they must cooperate with each other to get useful work done. In our second article, Robert Scott goes on lo describe some techniques for making sometimes selfish tasks play nice together.

Related to real-time control is a special kind of microcontroller: the PLC, or programmable logic controller. Francis Lyn walks us through some of the history behind PLCs and gives examples of one system available today.

Now that you've collected a large amount of data with your embedded controller, what better way to save some space than to compress that data? You've probably heard the names Lempel, Ziv, and Welch (what ever happened to Smith and Jones?); now you can find out what they did for compression algorithms. Dwayne Philips describes how the LZW compression method works and illustrates the algorithm with some working code.

Enough with the software. Let's get down to the chip level again with a comparison of two data logger designs: one using an 80C31 and the other a Dallas Semicondudor DS5000. John Dybowski covers all the bases, detailing how he made **his component choices in the two designs.**

In our columns, Ed wvers yet another HCS II module: the LCD-Link. Now the HCS has a way to interact with humans on the fly. Jeff digs up an old Circuit Cellar article and explores the use of simulators in checking the characteristics of a design. Tom pulls out another industry buzzword—multimedia--and looks at what's happening in the arena with his usual skepticism. Finally, John Dybowski (hmm, that name keeps popping up) describes how to make your embedded system more bulletproof through the proper use of watchdog timers.

We have lots of signal processing articles in the roundup for the next issue, plus we'll be talking about embedded interfacing, so stick around.

Ken

CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

OUNDER/EDITORIAL DIRECTOR
teve Ciarcia

IANAGING EDITOR
en Davidson

SSOCIATE EDITOR
isa Nadile

NGINEERING STAFF
leff Bachiochi & Ed Nisley

ONTRIBUTING EDITORS
om Cantrell & Chris Ciarcia

NEW PRODUCTS EDITOR
Harv Weiner

ART DIRECTOR
Lisa Ferry

STAFF RESEARCHERS:
Northeast
John Dybowski
Midwest
Jon Elson & Tim McDonough
West Coast
Frank Kuechmann

Cover Illustration by Robert Tinney

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 (ISSN 0896-8985) is published bimonthly by Circuit Cellar Incorporated, 4 Park Street, Suite 20, Vernon, CT 66666 (203) 6752751. Second-class postage paid at Vernon CT and additional offices. One-year (6 issues) subscription rate U.S.A. and possessions $17.95 Canada/Mexico $21.95, all other countries $32.95 All subscription orders payable in U.S. funds only via international postal money order or check drawr on U.S. bank. Direct subscription orders lo Circui Cellar INK, Subscriptions, P.O. Box 3050-C, Southeastern, PA 19396 or call (215) 630-1914.

POSTMASTER: Please send address change to Circuit Cellar INK. Circulation Dept., P.O. Bo: 3050-C, Southeastern, PA 193%.

HAJAR ASSOCIATES NATIONAL ADVERTISING REPRESENTATIVES

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

MID-ATLANTIC
Barbara Best
(908) 741-7744
Fax: (908) 741-6623

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

MIDWEST
Nanette Traetow
(706) 789-3080
Fax: (706) 789-3082

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

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

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

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

Entire contents copyright ©1992 by Circuit Cellar Incorporated. All rights reserved. Reproduction of thi publication in whole or in part without written consent from Circuit Cellar Inc. is prohibited.

INSIDE ISSUE 27

14 **Build a Real-Time Multitasking Executive**
by Mike Podanoffsky

22 **Resource Management in Cooperative Multitasking**
by Robert Scott

28 **Using Programmable Logic Controllers**
by Francis Lyn

36 **LZW Data Compression**
by Dwayne Phillips

SPECIAL SECTION Embedded Sensors & Storage

50 **The Elements of a Data Logger**
by John Dybowski

2 **Editor's** INK/Ken Davidson
Computing in Real Time

4 **Reader's** INK-Letters to the Editor

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

68 **Firmware Furnace/Ed** Nisley
An HCS II LCD Terminal

76 **From the Bench/Jeff Bachiochi**
Computers on the Brain (revisited)

84 **Silicon Update**/Tom Cantrell
Multimedia Madness/Couch Potato Computing

92 **Practical Algorithms/John Dybcwski**
Use a Watchdog to Keep Your Controller in Line

99 **ConnecTlme**—Excerpts from the Circuit Cellar BBS
conducted by Ken Davidson

112 **Steve's Own INK/Steve** Ciarcia
Making "Sense" of the World

97 **Advertiser's Index**

READER’S INK

Watch That Stack

I enjoyed Peter Hiscocks’s article, “State Machines in Software” *(Computer Applications Journal,* issue #26}. I’m glad to see a few more non-8031 articles.

For some years, I’ve used several of the techniques Peter mentions-both the constructed JMP and the “push and RTS hack.” He omitted one detail in the latter, however. The 6502 increments the return address when an RTS is performed. Consequently, the address pushed should be address *minus* 1.

The following fragment is an example:

```
        LDA  ROUTINE   ; Which subroutine wanted?
        ASL            ; Multiply by 2
        TAX            ; Use as offset
        LDA  TABLE,X
        PHA            ; Push hi byte of address
        DEX
        LDA  TABLE,X
        PHA            ; Push lo byte
        RTS            ; Jump to routine

                       ; Addr are lo byte, hi byte
TABLE  EOU  *
        DA   FIRSTROUTINE-  ; Address minus one
        DA   SECONDROUTINE-
        DA   THIRDROUTINE-
```

Thanks for a great magazine.

James Brodsky, Shell Beach, CA

RF People Tracker

I read with great interest Steve’s IR-Link article [Computer Applications *Journal,* issue #26}, especially the badge reader and the people-tracking section. Personal identification by IR is great, but as you mentioned there is a problem in aiming the IR source at the receiver. Increasing IR power might solve the problem but raises another one: powering IR LEDs at 4 watts is a lot power to get from batteries.

I think you should consider other solutions instead of IR, like RF. Radio-Command (RC), such as that used with toys or garage door openers, might be just what you need. ICs already exist from Motorola or National Semiconductor that don’t require anywhere near the power.

There is also Dallas Semiconductor, which has ICs related to identification and security with their Touch Memory and Proximity Memory. I never experimented with either, but recently I received a booklet titled 50 *Ways to Touch Memory,* and I think those components have great potential. If you ever ask for information, be sure to get this booklet. It shows many interesting applications with Touch Memories.

Jocelyn Lacombe, Laval, Quebec

[George *Martin, a member of the Circuit Cellar Design Team, responds]*

Close only counts in horse shoes and dancing.

We’ve done a lot of experimenting with the Dallas ICs. Recently, we worked on an assignment to construct a people tracking and locating system for a six-billion-dollar man’s new digs.

The Dallas system consists of a buried loop antenna, approximately 18″in diameter (transmitter in kHz range), a simple dipole antenna of 24”overall length (receiver in MHz range), and a battery-powered ID key (receive = kHz and transmit = MHz) that individuals were to carry on their person.

The system worked well, just not suitable for what we had in mind. The rooms were large (5O’by 50’) and we needed to locate people within a 5′ diameter circle.

As delivered, the buried loop was too powerful, sending signals out to a distance of 30′. It looked like a simple exercise to reduce the range of operation. We made the transmit loop smaller and reduced its power. But then the orientation of the portable tag became significant. The orientation would affect whether the tag recognized commands. The antenna in the tag was a ferrite loop similar to those in portable radios. What we created was a hand-held device that was orientation dependent (a lot like a hand-held IR unit). GREAT.

So we tried *a different type of antenna in the tag. No good.*

Then, we looked at a software scheme to help eliminate the directionality of the tag. Not even Bill Gates could have afforded those routines.

Next, we looked at a combination that included optical along with the Dallas ICs. One step away from the B1 bomber’s inertial navigation system.

Well, we went back to the beginning and put antennae in doorways and other passageways. The *people had to pass over the antennae and their movement would be recognized. The location goal of 5′ has been shelved for the present.*

The Dallas part worked fine and we came close, but....

Trainable IR-Link

I applaud the design of the HCS II. Its networked approach seems quite revolutionary for a domestic product. Regarding home use, however, the IR-Link sounds fine for security or tracking, but have you guys forgotten that most of us use IR for TV and VCR remote controls? Any chance of putting that function in the HCS II?

Dan Draper, Washington, D.C.

I *must apologize because we did in fact forget* trainable *IR remotes when designing the IR-Link. Perhaps because tracking and security were so much an issue at the time (see George Martin's letter above), I was blinded by the IR (pun intended). Fortunately, because so many of you communicate with the Circuit Cellar staff on the BBS, this fact did not go unnoticed for long. I extend special thanks to Stan Eker in particular for suggesting a plausible solution to us as well.*

The net result is that a little "Nisley Magic" in the software will soon allow the original IR-Link hardware to function as a trainable IR remote control that sends commands to TVs, VCRs, and so forth, under the HCS program control. The new trainable unit will be designated as the MCIR-Link. There will also be a software upgrade for original IR-Link purchasers.

Steve

We Want to Hear from You

Our readers are encouraged 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

REAL TIME
Multi-Tasking Operating System

- Task Manager
- List Manager
- Flag Manager
- Resource & Semaphore Manager
- ROMable
- User Conflgure-able
- All CMX Functions Contained in a Library
- Easily Interfaces to Assembly Language
- Event Manager
- Timed Procedure Manager
- Message Manager
- Memory Manager
- Supports Nested Interrupts
- All Functions Written in "C" for Portability
- CMX Task Scheduler, Timer Task, and Interrupt Handler Written in Assembly for Speed and Optimization

8051 and Derivatives	8096/196
Hitachi H8/300 Series	68HC11/16
Other Microprocessors	Z80/64180

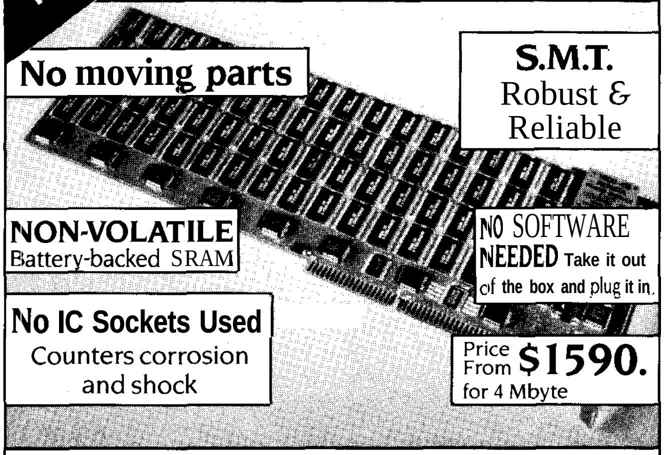
Source Code   No Royalties

CMX COMPANY

Call For More Information.
508-872-7675
19 Indian Head Heights
Framingham, MA 0 1701

#102

NEW SRAM Solid State DISK



No moving parts

S.M.T. Robust & Reliable

NON-VOLATILE
Battery-backed SRAM

NO SOFTWARE NEEDED Take it out of the box and plug it in.

No IC Sockets Used
Counters corrosion and shock

Price From $1590. for 4 Mbyte

IDEAL FOR HARSH ENVIRONMENTS
Humidity, dust, vibration, shock

- 4 to 64 Mbytes
- Standard AT ISA bus
- Fully DOS Compatible
- Seek time, less than 0.2 millisecs
- Data retention at least 2 years
- Device driver on-board

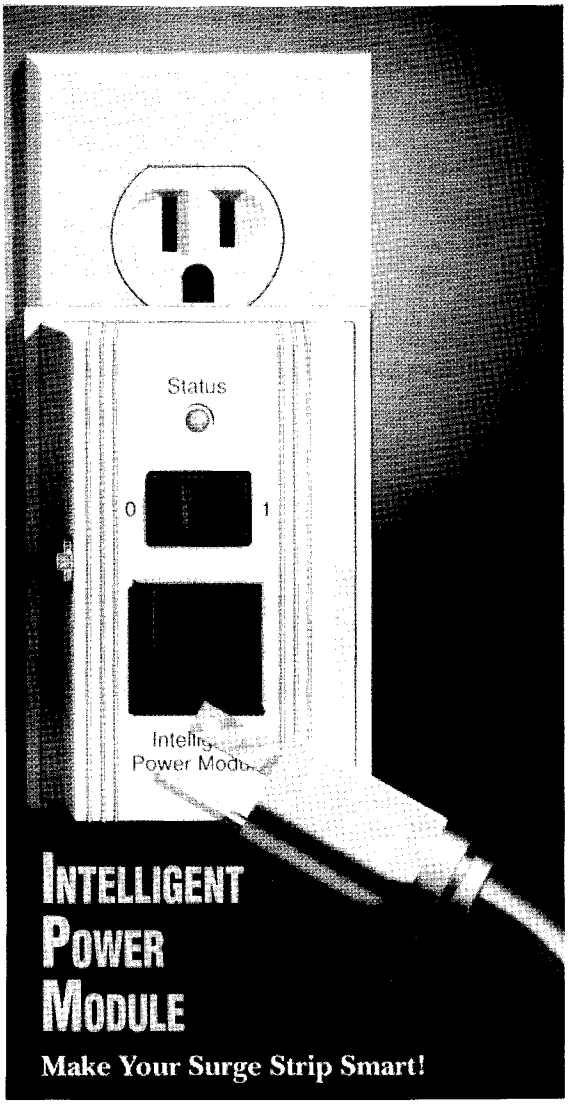
Business Machine Interfaces Inc.
2 13-05 39th Ave., Bayside NY 11361

Canada (4 16) 756-9477
FAX (416) 756-4118
U.S. 1-800-663-4605

!103

NEW PRODUCT NEWS

Edited by Harv Weiner



LOW-COST INTELLIGENT POWER MODULE

Unattended PCs that run long jobs during the night no longer have to keep their power on after their tasks are finished. Server Technology has announced a computer-activated power module that connects between a wall outlet and power strip to allow a PC to control AC power.

The **Intelligent Power Module** automatically powers off a PC when its jobs are completed, and eliminates the need to run a PC 24 hours a day. Some of the benefits include reducing the risks of unauthorized use or storm-related voltage and electrical damage, reduced operational time, reduced energy costs, and reduced equipment wear.

The Intelligent Power Module is activated by receiving ON or OFF signals from a PC via a signal cable connected to its serial or parallel port. By placing the power-off command at the end of a batch file used to run unattended operation, the PC will send a power-off signal to the Module when all jobs are complete, shutting down the system. This feature can be extended to a power strip for the control of a wide variety of office, data collection, laboratory, and manufacturing equipment.

A master PC with multiple serial ports controlling multiple Modules can control power flow to other types of equipment located throughout the office. The master PC running scheduling software can execute power-on or power-off commands at any time of the day, signalling the power modules. Schedule equipment requiring a lengthy warm-up period for early morning power on and be ready for operation when the office opens for business.

The Intelligent Power Module comes complete with software commands on diskette and a signal cable to link a PC to the Module. The suggested retail price is **$59.**

Server Technology, Inc.
2332-B Walsh Ave.
Santa Clara, CA 95051
(408) 988-0142
Fax: (408) 988-0992

#500

REMOTE VGA OPERATION

Serial Video+ allows the remote operation of a monochrome or color VGA monitor from any serial port of any computer. You can connect as many monitors as serial ports, greatly expanding the one- or two-monitor capacity of most systems.

For example, an IBM AT with a special eight-port serial board would allow nine monitors to be connected. Microcontroller applications are also simplified because the video circuitry normally included to drive a small LCD display can now be replaced by a simple serial port.

The same remote monitor being driven from the single serial port can also be equipped with a touch screen to expend terminal capability. Applications for this device include point of sale systems, activity calendars in hotels and convention centers, home automation centers, remote data logging terminals and status centers, and so forth. Pricing was not available at press time.

IntelliHome Controls
Attn.: Serial Video+
1719 Bank St., Ste 301
Ottawa, Ontario K1V 7Z4
(613) 731-1566
Fax: (613) 737-3052
#501

NEW PRODUCT NEWS

CUSTOM DATA LOGGER

For those situations where you need a custom data logger to store large amounts of nonvolatile data, The Saelig Company offers a low-cost solution. The **TDS2020C Data Logger Module** offers 8 MB or more of removable flash or RAM card memory storage, and its 4" x 3" size can be built into your own product. The TDS2020C consists of the TDS2020 computer and the TDS2020CM Card Memory board.

The TDS2020 computer uses the Hitachi H8/532 microprocessor and has on board 16K of FORTH as well as a full symbolic assembler. Programs can be written in high-level language, mixing with assembler if required. Up to 45K of memory is available for compiled programs, and up to 512K bytes of RAM, EEPROM, or Flash memory to keep vital data while the board is not working.

Both alphanumeric and graphics LCDs connect directly, and the built-in software can scan a keyboard with a matrix up to 8 x 8. The board has between 26 and 41 parallel I/O lines [depending on options selected) and two serial ports. The ADC has eight channels of 10-bit resolution (better than 1 in 1000) and the DAC has three channels of 8-bit (1 in 250).

Additional features include four hardware counter-timers, I²C bus, two separate watchdog timers, nonvolatile time-of-day clock, and multitasking.

The TDS2020CM sits on top of the computer to form a sandwich. Up to two TDS2020CM boards, each with Card Memories of up to 4 MB, can be included. Card memories are accessed using a 32-bit address, and software is supplied on disk to allow a write or read to any byte or 16-bit word.

The data can be organized as appropriate to the application. For example, in scanning eight 10-bit A/D channels, each can be allocated 2 bytes, and a single byte will suffice for every eight digital channels being logged.

Data can be recovered from a Card Memory through either the TDS2020 or a PC-compatible. For example, use the TDS2020C module in the field to collect measurements, bring the Card Memory back to base and read it with another such module, or use the remote computer to send the data periodically over a telephone line.

The TDS2020C sells for $372 in small quantities.

The Saelig Company
1193 Moseley Rd.
Victor, NY 14564
(716) 425-3753
Fax: (716) 425-3835
#502

STEPPER MOTOR CONTROL SYSTEM

A stepper motor control system that can be driven from the parallel port of a PC or compatible is available from MAS Electronics. The **CSTEP#2 System** consists of a 3.75" x **2.75"** printed circuit interface and driver board, an interface cable, and demonstration software.

The interface and driver board will operate two 5-volt to 24-VDC stepper motors with up to 1.5 amperes per phase drive current. The interface cable plugs into the standard parallel port of any PC-compatible computer. The user friendly, menu-driven demonstration software allows operation of the motors in full or half steps, with options for single stepping, variable speed continuous rotation, directional control, and motion of a given number of steps. The software will run on any PC-compatible computer. Software and interface cables are also available for the user port on Commodore 64 and 128 computers.

A major advantage of the CSTEP#2 System is its ease of setup and use. A motion control demonstration can be quickly set up and run with the CSTEP#2 System, a personal or laptop computer, one or two motors, and a DC power supply. The System can be used to experiment and develop applications for stepper motors. For portable applications with a laptop computer, the DC power supply can be a battery pack.

The CSTEP#2 System for 5-volt to 24-VDC motors is priced at $90 including software and interface cable.

MAS Electronics
931 Lincoln Rd. • Birdsboro, PA 19508-1801 • (215) 582-4864

#503

NEW PRODUCT NEWS

DATA ACQUISITION BOARD TUTORIAL

ADAC Corporation has created a novel method to change the way you feel about data acquisition boards. Featuring ADAC's family of Direct Connect data acquisition boards, Direct View software eliminates long, painful, and costly learning curves, and makes "out of the box and running in minutes" a reality.

Supplied on a single floppy with no need for a user's manual, Direct View is a complete I/O tutorial on a diskette. The tutorial makes connecting transducer wiring and understanding signal conditioning easy. The friendly, mouse-driven interface guides you through board jumper settings, with help screens to explain the function of each jumper. Problems setting the board's address, A/D input range, interrupt level, and so forth, are eliminated.

Direct View guides you through all selectable board options including board address, DMA channel, and interrupt level. Graphic representations of the Direct Connect data acquisition boards includes online help screens fully explaining the function controlled by each jumper option. By simply clicking on any board jumper shown in Direct View, you can move it in software, and be shown the resulting impact on the board functionality. A summary page showing you selected options provides a convenient board configuration overview.

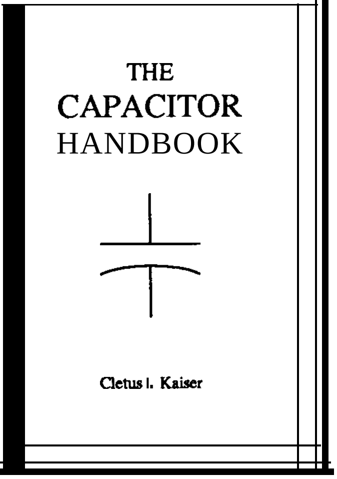
Direct View's unique menu-driven tutorials explain the proper use of thermocouples, strain gauges, and RTDs. Running program examples for GW-BASIC, Turbo Pascal, and Turbo C are included. In addition, Direct View includes block diagrams explaining the proper connection of field wiring to the I/O board.

Once the I/O board has been configured for your application, Direct View can be used to perform data acquisition on all channels, expressing the data as counts, volts, and temperature or strain where applicable. It also performs real-time display and storage of data to an ASCII file. Direct View is free of charge.

ADAC Corporation
70 Tower Office Park • Woburn, MA 01801
(617) 935-6668 . Fax: (617) 938-6553

#504

CAPACITOR HANDBOOK



THE
CAPACITOR
HANDBOOK

Cletus I. Kaiser

The Capacitor Handbook has been published by CJ Publishing. Written by Cletus J. Kaiser, the handbook begins with a general introduction to capacitors. It includes chapters covering most of the popular kinds of capacitor construction, including ceramic, plastic film, aluminum electrolytic, tantalum, glass, and mica.

Appendices in the handbook include a useful capacitor selection guide that summarizes key characteristics of each kind of device, basic formulae, and often-used symbols. A glossary, bibliography, and index round out the handbook

The Capacitor Handbook contains 126 pages and is available for $14.95 plus $3.50 shipping.

CJ Publishing
2851 W. 127th St.
Olathe, KS 66061
(913) 764-3577
Fax: (913) 764-8909

#505

NEW PRODUCT NEWS

XT/AT-BASED FUNCTION GENERATOR AND FREQUENCY COUNTER

StarPC Instruments is offering a low-cost XT/AT-based Function Generator and Frequency Counter for stand-alone, automatic test, or data acquisition applications. The **OscPC-100** features include a programmable timed sweep, a logarithmic sweep, and burst output modes of operation. The frequency range is 1 Hz to 25 kHz with 16-bit resolution accurate to 0.005%. The level is programmable from 0 to 10 volts peak to peak in 20-mV increments. The DC to 10-MHz Frequency Counter can detect pulses as narrow as 12 ns and has a maximum count error of 1 Hz or 5 ppm. The 5-Hz to l-MHz Pulse output includes a time-programmable pulse at a programmable rate; the pulse width may be programmed in one-sixth of a microsecond increments.

The OscPC consists of a half-length card with dual BNC I/O connectors. The software runs in either command line or menu modes and requires no system resources, such as memory or interrupts, once the hardware has been set. Sweep or Burst may be terminated automatically after a preset number of repeats or manually by pressing ESC.

A Software Link Library with a variety of direct call functions is provided for easier interface in automatic test of data acquisition systems. The libraries include the object code of the major frequency generation and measurement functions.

The **OscPC-200** high-performance version can count pulses for frequencies up to 12.0 MHz with a maximum error of 0.00025% (2.5 ppm). The pulse signal reaches a maximum frequency of 2.0 MHz and the timing pulse may be programmed in one-twelfth of a microsecond step.

The OscPC- 100 retails for $180 and the OscPC-20 for $200.

StarPC Instruments
P.O. Box 84418
Sunnyvale, CA 94086-4418
(408) 739-5 117

#506

smx simple multitasking executive v2.1

FULL FEATURED KERNEL

□ task manager
□ intertask comm.
□ i/o, events, & timing
□ preemptive
□ memory manager
□ error manager
□ resource manager
cl ROM'able

EASY TO USE

□ libraries for Microsoft C,Borland C/C++, Zortech C/C++ and assembler
□ Quick Start, User's Guide, and Reference manuals
□ standalone, PC, and DOS platforms
□ *smxProbe* task debugger
□ 6 months free support and updates

FAST & SMALL

□ 15 usec max. interrupt latency
□ 150 usec typical task switch
□ 8 to 25 KB code size

Ask about our $95 evaluation kit.

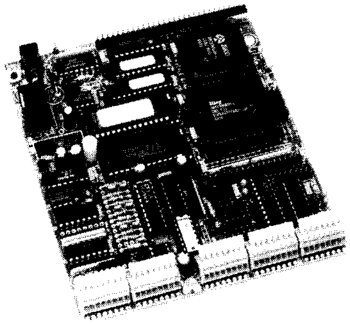
MICRO pd DIGITAL
6402 Tulagi St.
Cypress CA 90630-5630

Call for information:
1-800-366-2491
FAX 714-891-2363

#105

MICROCONTROLLERS

- C Programmable
- Data Acquisition
- Control / Test
- Excellent Support
- From $159 Qty 1
- *New* Keyboard Display Modules



Use our Little Giant™ and Tiny Giant™ miniature controllers to computerize your product, plant or test department. Features built-in power supply, digital I/O to 48+ lines, serial I/O (RS232 / RS485), A/D converters to 20 bits, solenoid drivers, time of day clock, battery backed memory, watchdog, field wiring connectors, up to 8 X 40 LCD with graphics, and more! Our $195 interactive Dynamic C™ makes serious software development easy. You're only one phone call away from a total solution.

Z-World Engineering
1724 Picasso Ave., Davis, CA 95616
(916) 757-3737 Fax: (916) 753-5141
Automatic Fax: (916) 753-0618
(Call from your fax, request catalog #18)

#106

NEW PRODUCT NEWS

WINDOWS-BASED METER DISPLAY SOFTWARE

Attractive visual display of process values has been simplified by new software introduced by Intelligent Instrumentation/Burr-Brown. PCImeter is a graphic meter software package specifically designed for the Microsoft Windows environment.

Running under Windows 3.0 or higher, PCImeter acquires analog data for display as on-screen digital and analog meters and bar graphs. High and low alarm setpoints can be visually displayed in each meter style. In addition, each meter can be configured to trigger digital outputs when setpoints are reached.

Process values can be modified to engineering units using scaling factors, and linearization for thermocouples is also supported. Meters can be saved to PCImeter application files and can be reconfigured at any time.

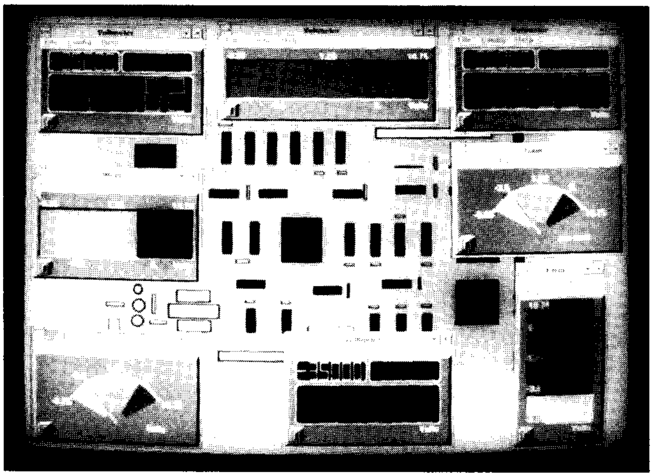
Up to 16 meters can run simultaneously. Using Windows' Dynamic Data Exchange (DDE) feature, PCImeter can also export data to Excel spreadsheets for analysis and report generation. Dynamic Link Libraries (DLLs), or hardware drivers for the PCI200998C series Multifunction I/O boards, are included with PCImeter software. Additional DLL drivers are currently under development.

PCImeter runs on IBM PC/AT-compatibles and EISA personal computers with a hard disk and floppy drive. Other hardware requirements include 1 MB or more of RAM, monochrome graphics, an EGA or VGA graphics monitor, and an Intelligent Instrumentation I/O board.

PCImeter (PCI-20365S-1) sells for $95. A free demo diskette in either 5.25" or 3.5" formats is also available.

Intelligent Instrumentation
1141 W. **Grant Rd., MS 131 . Tucson, AZ 85705**
(602) 623-9801 . Fax: **(602) 623-8965**

#507



UNIVERSAL INTEGRATED DEVELOPMENT/ COMMUNICATIONS ENVIRONMENT

Life Force Technology has introduced Armadillo+ for the IBM PC and its compatibles. Armadillo+ combines the power of editors, cross-assemblers, compilers, disassemblers, simulators, and data conversion utilities into an ergonomic serial environment. The environment is completely menu driven, utilizing pull-down menus with mouse support typical of those found in Microsoft language environments. Armadillo+ supports all families of cross-assemblers, cross-compilers, disassemblers, and simulators.

Armadillo+ enables you to edit a source file, assemble it, upload it to a target microcontroller board or other microprocessor-based system, and debug all without having to quit the communications program. This process eliminates the need to manually type lengthy, cryptic, and time consuming command lines for each of the applications associated with traditional microcontroller cross-development methods. Once configured to your needs, each of the above tasks is as simple as pressing a couple of keys or pointing and clicking a mouse. After each task is completed, you are automatically returned to communications with the target microcontroller board.

Armadillo+ supports all families of cross-assemblers and features user-definable Utilities menus for running various development applications associated with a particular project. Serial communications support includes COM1-COM4; choices of 5, 6, 7, or 8 data bits; even, odd, or no parity; 1, 1.5, or 2 stop bits; XON/ XOFF, CTS/DTR, or both types of handshaking; line feed; control character; and high-bit filtering. The vendor has proven that Armadillo+ can keep up with a continuous serial input at 9600 bps on a 16-MHz '386 system with no handshaking. Video support includes CGA, monochrome, Hercules, EGA, VGA, and LCD.

Armadillo+ (an acronym for Asynchronous Responsive MultiAssembler Development Integrated Link to Logical Operation) sells for $99.

Life Force Technology
5477 Rutledge Rd.
Virginia Beach, VA 23484
(804) 479-3893

#508

FEATURES

14 Build a Real-Time Multitasking Executive

22 Resource Management in Cooperative Multitasking

28 Using Programmable Logic Controllers

36 LZW Data Compression

Build a Real-Time Multitasking Executive

Having several independent, concurrent tasks running on a real-time system often simplifies program development. Find out what it takes to put such a system together.

FEATURE ARTICLE

Mike Podanoffsky

Oelephone switching, medical monitoring, unmanned spacecraft, and home control systems are a few good examples of real-time multitasking systems in action. These systems all must respond to multiple, and at times contradicting, events. Developing the software for them would be impossibly complex without some task management, which at the very least sorts priorities.

A real-time multitasking executive compartmentalizes tasks, events, and responses that break down complex problems into specific, specialized, and responsive code.

A multitasking system executes multiple tasks concurrently. A system is real-time if it is able to run tasks in a timely response to real events. The executive is the code that controls which tasks should be run, suspended, and given higher priority.

Consider a state-of-the-art home appliance control system as an example of a real-time multitasker. This system acts around activities that occur periodically throughout the day including turning appliances and lights on and off, monitoring doors and windows for traffic, monitoring and arming fire, smoke, or motion detectors, and supporting a control panel or telephone response to accept programming changes. Also, autodial and alarm functions to police, fire, and emergency services are needed. These examples all represent real-time events.

The real-time system would contain a task responsible for monitoring alarms, another for turning lights on and off, another for telephone response, and so forth. The job of each task is narrow, focused, and independent of the remainder of the system. The real-time executive determines which task to run in response to which events.

I once used this real-time system to capture and distribute Associated Press (AP) newswire to newspapers. The wire service is transmitted by AP to subscribing newspapers (yes, they charge for it, and charge far more than you'd want to pay for home newswire delivery). Each news story is transmitted with a unique identifier as to subject, content, length, format, edition, and urgency. The wire has no start or stop; your system has to be ready to receive the copy or the story is lost. Eventually, your PC will contain repeat and obsolete stories requiring deletion.

An Interrupt Service Routine (ISR) physically serviced the serial data communications received and buffered the data in memory. A task waited for data, received events, then "read" the news story headers to determine where the story should be stored. Another task snooped for obsolete stories and remove them from the system. The "remove" task only ran at certain low-peak times, which allowed wire capture to be as efficient and timely as possible. The wire-service program not only multitasked but ran on an IBM PC with MS-DOS, so I was able to receive the benefits of both.

Apart from the real-time executive itself, the demo program I wrote and describe a bit later shows several tasks running at the same time, utilizing the DOS file system. I still use my 12-MHz '286 machine, and the performance of the real-time executive on that platform is very good.

Listing 1—*Tasks are, in general, simply standard functions. Their names are passed to he multitasking kernel so the kernel knows how to invoke them.*

```
/* ///////////////////////////////////////////////////////

    any_task()
   - -   - - - - - - - - - - - - - - - - - - - - - - - - -

    a task is written as just a C function
     or an ASM subroutine.

/////////////////////////////////////////////////////// */

void any_task(void far * argument )

    int chars-read:
    FILE far * file;

    file = fopen(argument, "ra");

    while ( file ){
        .
        .
        .
    }
}
```

For those of you who already know a bit about multitasking systems, the software offered here supports multiple tasks, preemptive scheduling, time slicing between tasks, differing priorities for tasks, dynamically alterable priorities, watchdog timers, and time-triggered tasks. Source code is provided, of course, in C and in assembly language. The code has been tested on Microsoft c 5.1.

BASIC REAL-TIME CONCEPTS

A multitasking system gives the perception of running multiple tasks at the same time. For example, you could process multiple telephone data transfers in one task while performing text editing in another. In fact, only a single task is running at any given moment. Because the real-time executive responds to events, it performs a task switch as events mandate, which happens fast enough to make the system appear to run many tasks concurrently.

A task is a logically independent piece of code that performs a job when activated by an event. Events can be many things including a key press, a character received, a signal detected, or a clock ticking. After a task processes the event, it returns to waiting for another. Other tasks are then given a chance to run, and they in turn receive and process events.

Tasks have priorities. The one with the highest priority runs first, followed consecutively by tasks of lower priorities. Tasks may be preempted. If an event occurs for a task of higher priority, the lower priority task is interrupted and the higher priority task is given preferential treatment. Such a setup is known as preemptive scheduling.

Not all multitasking systems allow preemption. Some systems support only what is known as cooperative multitasking [e.g., Microsoft Windows). Under cooperative multitasking, the currently running task must periodically voluntarily yield to permit other tasks a chance to run. Without preemptive scheduling, a low priority task can completely take over a system by never yielding.

Voluntary yielding happens more often than you'd suspect. Every time a task waits for input (or any other event], it is voluntarily yielding. Preemptive scheduling completely eliminates the need for a task to concern itself with overall system performance. It also allows the multitasker to support task time slicing. A time slice is a period of time given to a task in which to run. When the time slice expires, a check is made to determine if other tasks of the same or higher priority are ready to run and if a task switch is required.

Related to time slice is sleep. Sleep is, of course, a period of time during which a task doesn't run. Tasks may sleep for specific periods or until a specific time of the day. Surprisingly, there are cases where you want to perform a function only every few seconds or minutes.

Let me use a network-based application I once wrote as an example. If the network was not responding, I would store all records input on a local disk. Once the network was up, the saved transaction file would be uploaded. Constantly checking for network availability made no sense. In fact, checking the network every 30 seconds was more than sufficient.

Using this capability makes tremendous sense with a multitasking system, but it could be problematic with a more traditional program. In my specific application, the data entry task ran separate from the database-updating or network-monitoring tasks. The nuances of update speed or network loading were removed from the data entry task and the user entering data. The physical update happened in the background.

In order to be able to perform any task switching at all, each task must maintain its own independent stack, which may be an obvious point. The stack basically contains the thread of your code. Each function call's return address is pushed onto the stack, as are other data, including pushes and temporary variable allocation. When the task is restarted from an interruption, the stack must be in exactly the same condition in order for execution to resume, which means not sharing the stack.

THE MULTITASKING SYSTEM

Let me now discuss the multitasker. My intent is to describe the architecture of the real-time executive. The multitasker is written in C except for some interrupt service and stack manipulation routines, which are written in assembly language. I have also made an all-assembler version available with the downloadable source code if you are more comfortable with this language. I also include in the source code a usage guide for each function within the real-time executive.

Listing 2—*Typical system start-up code initializes timers and defines events and tasks for the multitasking kernel. Each task is referenced by name and is assigned, among other things, a stack size and a priority.*

```
/* ////////////////////////////////////////////////////////////////
   main0


   a) initialize screen.
   b) initialize tasking system.
   c) create tasks.
   d) let scheduler start after initialization.

//////////////////////////////////////////////////////////////// */
main0
{
    initTaskSystem();

    setTimer(ONE_SECOND. one_second_interval. NULL);

    defineEventFct(KEYBOARD_WAIT. &keyboardEventFct. NULL);

    defineTask(monitor_tasks. NO-ARGUMENT. 8192. 1024. "name1");
    defineTask(typeout_file.  &rect2.      8192. 96,   "name2");
    defineTask(typeout_file.  &rect3.      8192. 96.   "name3");
    defineTask(task_4.        NO-ARGUMENT. 8192. 96.   "name4");

    scheduler-O:
}
```

TASKS

A task is code that performs some work. It has a current execution address, a starting address, a stack, a priority, and other variables pertinent to execution. These parameters are maintained by the real-time executive in a *tasks* data structure. This task data is maintained in the *tusks* array, which is made up of entries containing the task data for a respective task.

The order of the entries in the *tasks* array is inconsequential and is allocated on a first come, first served basis. The order bears no relationship to task priorities. The maximum number of allocated tasks is customizable, as I'll describe later.

The index into the *tasks* array is called the task ID. The pointer to the task entry in the *tasks* array is called the task pointer. It is not the task's execution address, but it is the pointer to a structure of information that the real-time executive maintains for the task.

In order for a task to be managed by the real-time executive, the entry in the *tusks* array must be initialized using the defineTask() function. The task does not exist without calling defineTask(). You must pass the starting address of the task, the task's priority, and maximum stack size. A stack will automatically be created for your task.

Optionally, you may pass a single far pointer argument to the task, and you may assign your task a name. You are not limited by the amount or type of memory you pass to a task because the argument you pass can be a pointer to a structure of information anywhere in memory. Task names are useful for both debugging and intertask communications.

From a programming point of view, a task appears as any C function. Listings 1 and 2 show both sample task code and the system start-up code. I provide additional programming information for the real-time executive in the separate guide available with the source code. A task may be written in any computer language.

The real-time executive organizes tasks in priority order. Rather than rearrange the order of the tasks in the *tusks* array, which would invalidate any task pointer or task ID, the real-time executive maintains a separate *priorities* array. The *priorities* array maintains the priority and task ID for each task in the *tasks* array in high-to-low priority order, or in other words, in inverse priority order. Each time the priority of a task is altered, the change is reflected in the *priorities* array.

EVENT FLAGS

Events are actions to which the real-time system is required to respond, and they are usually the result of a physical action [e.g., a key is pressed, data is received, or a phone rings). With the right detectors, the opening of doors, temperature changes, and even verbal commands can be events.

Events may also be logical. Nothing prevents one task from signaling another by setting or clearing event flags. In my database problem described above, one of the events was a no-network detection. An expensive view of events is best; use an event to its fullest advantage.

The real-time executive's job is to find the task responsible for supporting an event. Tasks inform the real-time executive when they are waiting for an event using the waitEvent() function. You may set the current task [your task) or any other task to wait for an event. When the event occurs, the task waiting on the event is started. All events must be defined in the custom files and are assigned event ID numbers. The IDs should be assigned sequentially starting with 1.

A *system events* array, named SystemEvents, exists within the real-time executive. When an event occurs, it is flagged in SystemEvents. This array is a bit array, where a single bit represents an event. If the bit is set, then the event has occurred.

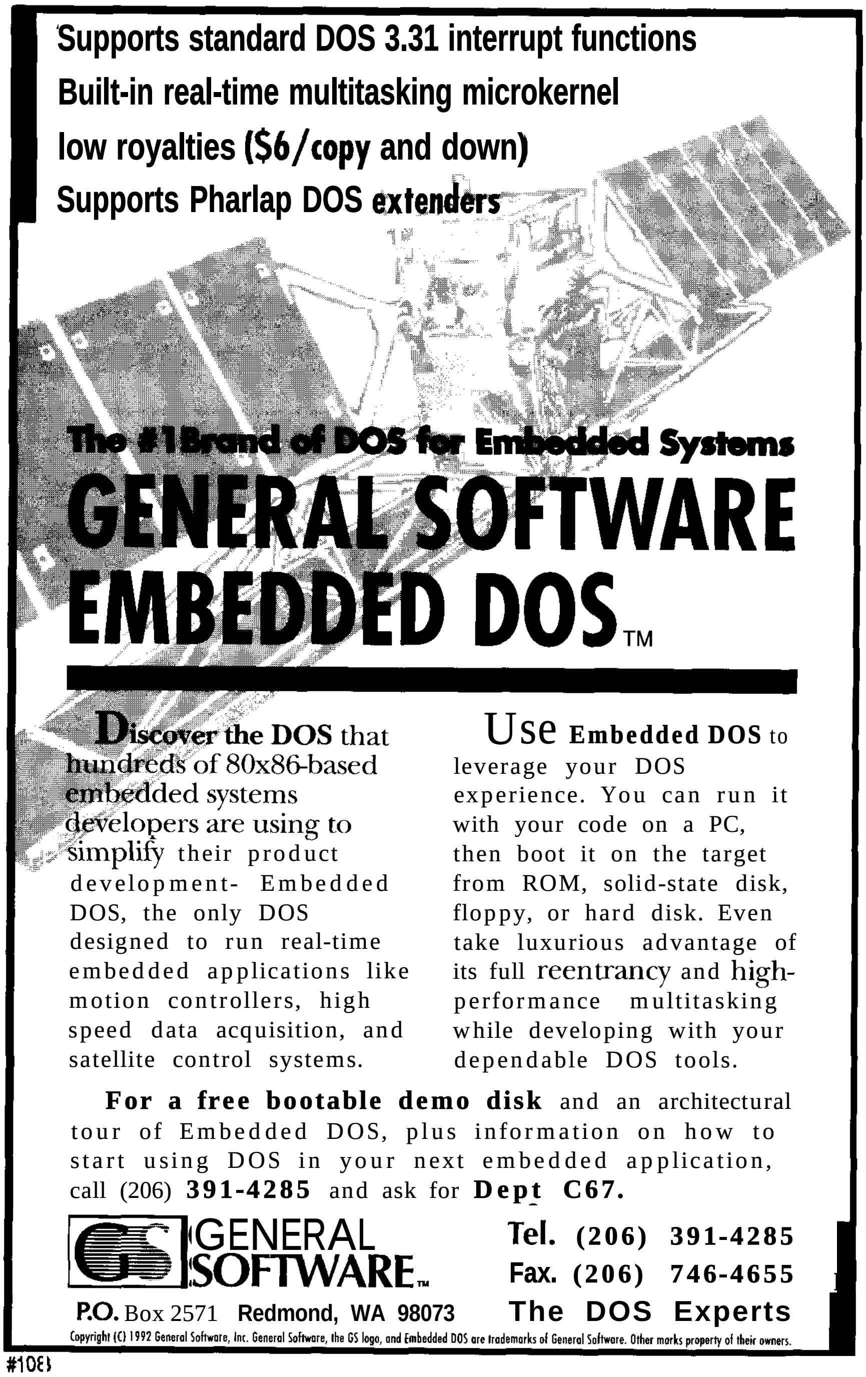
Within each task entry in the *tasks* array, there is also an *event* array. While the SystemEvents array represents events that have happened, the event array for the task entry represents those events waiting for this task.

The scheduler matches the SystemEvents with the waiting events to determine if the task can be restarted. The waiting bits for a task are set using the waitEvent() function. When waitEvent() is called, the waiting bit referenced is set and the task is suspended until the event occurs. Usually, SystemEvents bits are set by an ISR, some code like another task, or an event function (a very useful mechanism augmenting the SystemEvents). An event function need not exist for every event.

The keyboard or any buffered device is an excellent example of an event helped by an event function. When there is an event function present for a given event ID, the bit value in the SystemEvents array is ignored. The event function is called instead, and that value is considered the valid state of the event.

An example of a single event is a key is typed at a keyboard. The event bit in the SystemEvents **array** is set as expected by the keyboard ISR. As long as the task is able to keep up with the keyboard, each key typed follows the same pattern: the event flag is set, the task is activated to respond to the event, then the event flag in the Systemivents **array** is cleared.

When the keyboard gets ahead of the task, that is, when there are



Supports standard DOS 3.31 interrupt functions
Built-in real-time multitasking microkernel
low royalties ($6/copy and down)
Supports Pharlap DOS extenders

The #1 Brand of DOS for Embedded Systems
GENERAL SOFTWARE EMBEDDED DOS™

Discover the DOS that hundreds of 80x86-based embedded systems developers are using to simplify their product development- Embedded DOS, the only DOS designed to run real-time embedded applications like motion controllers, high speed data acquisition, and satellite control systems.

Use **Embedded DOS** to leverage your DOS experience. You can run it with your code on a PC, then boot it on the target from ROM, solid-state disk, floppy, or hard disk. Even take luxurious advantage of its full reentrancy and high-performance multitasking while developing with your dependable DOS tools.

For a free bootable demo disk and an architectural tour of Embedded DOS, plus information on how to start using DOS in your next embedded application, call (206) **391-4285** and ask for **Dept C67.**

GENERAL SOFTWARE™
P.O. Box 2571 Redmond, WA 98073

Tel. (206) 391-4285
Fax. (206) 746-4655
The DOS Experts

Copyright (C) 1992 General Software, Inc. General Software, the GS logo, and Embedded DOS are trademarks of General Software. Other marks property of their owners.

#108

several characters stored in the keyboard receive buffer, the pattern of clearing the keyboard SystemEvents bit will mask that additional characters are available. The SystemEvents flag will state there are no keyboard events left when in fact there are characters left to process. By interrogating the event function for the keyboard, the event function may return a "true" whenever there are characters to process.

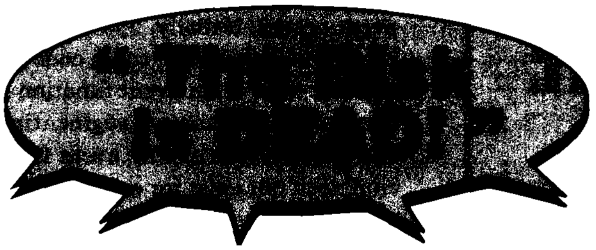
Event functions are an ideal mechanism for interfacing events and ISRs. SystemEvents are set by calling the setSystemEvent() function and can be set by any task, event function, or ISR.

TIMERS

The time slice, task sleep, and independent timers available within the real-time executive are interrelated. Several timers can be built into the real-time executive by changing the value of MAX_T I M ERS in the custom files. This simple adjustment allows you to have hundreds of timers.

Listing 3—*Int 09 is the keyboard interrupt. If a key is detected and saved by the ROM BIOS, it will force the setKbdEvent flag* *end* ***call the*** *scheduler* ***to evaluate task priorities.***

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
:  Int09()
;.........................................................;
;
Int09   proc   far

        SaveRegisters

        _Int    _ori gi nal KbdHdwTrap    ; do normal kbd duties
        mov     ax,40h
        mov     ds.ax                     ; look at bios kbd area
        mov     bx.1Ah

        cli
        mov     ax,word ptr Cbxl          ; 1Ah
        cmp     ax,word ptr [bx+2]        ; 1Ch
        sti
        jz      Int09_20                  ; if zero, no keys pending ->

        mov     ds, word ptr cs:[Rtx_DataSegment]
        call    rtx_text:setKbdEvent      ; say keyboard event occurred
        call    scheduler

Int09_20:
        RestoreRegisters
        iret

Int09   endp
```



The disk reliability problem is widespread.

There is an Alternative!

Dust, moisture and vibration kill rotating disks every day. **Instant Access**™ **No Motion**™ solid state memories offer extremely high reliability, light speed access and hermetically-sealed data integrity. **Instant Access** is available in FLASH EEPROM and NVRAM and is programmable in DOS or WINDOWS. Rely on **Instant** Access-the demonstrably superior No **Motion Memory.**

Mission Critical · **High Security** · **Banking** · **Military** · **Communications** · **Mobile** · **Diskless PC** · **Hazardous Environments**

PRODUCTIVITY ENHANCEMENT PRODUCTS, INC.

26072 Merit Circle, Ste. 110 . Laguna Hills, CA 92653
1(800) 451-DISK . FAX (714) 348-1310

#10

4 Configurations Available

The Heart of the Matter...

RTXC™

Real-Time Multitasking Executive

. HITACHI 6303 • INTEL 80x88/x86, 80x96, 80x51
. MOTOROLA 680x0, 683xx, 68HC11, 68HC16
. INMOS T400, T800 . ZILOG Z80/Z180

- **Preemptive Scheduling**
- **Fixed or Dynamic Priorities**
- **Timeout on some services**
- **Configurable and ROMable**
- **Intertask Communications**
  - **-Messages**
  - **-Queues**
  - **-Semaphores**
- **Memory Management**
- **Resource Manager**
- **Over 50 Executive Services Available**
- **System Level Debugging Utility**
- **System Generation Utility**
- **450 Page User Manual**
- **Written in C**
- **Source Code Included**
- **No Royalties**
- **Technical Support**
- **Broad C Compiler Support**
- **Sensible License Agreement**
- Most **Popular C Compilers supported**

Ask about ASSIST™ **Our new PC Development Package Combination**

One Time License Fee From $995
Discounts for Multiple Licenses/Ports
The only real-time kernel you'll ever need™

A.T. BARRETT & ASSOCIATES, INC.
11501 Chimney Rock, Houston, TX 77035
FAX 713/728-1049
Phone 800/525-4302 or 713/728-9688

FREE DEMO AVAILABLE

#1

A timer is given a wait value. When the required wait ticks have been reached, the timer calls a routine that you have specified and passes the argument that you have specified when the timer was armed. The argument passed may be a pointer to any structure of data. The function is referred to as a timer function and in no respect differs from any normal C function. A timer function may be written in any computer language. Please refer to the guide that comes with the source code.

Timers may automatically rearm themselves by recomputing the wait period. To make a timer a periodic timer, pass the PERIODIC argument in the function call.

Timers are supported by two functions: setTimer() and setTimer_idO. setTimer_idO will arm a specific timer, which will be known by its timer_id. If there are nine timers, then the timer IDs will be zero through eight. setTimer() will arm the first available timer. Timers may be cleared, extended, or shortened at any time by using the clearTimer() function or sending a new timer value to the timer.

Related to timers is the task sleep function, which is supported by the functions sleepTask() and sleepTill(). A sleeping task will not execute until its sleep period passes. Extending, contracting, or canceling the sleep period for a task is possible. Use the sleepTask() function and change its sleep period by extending, shortening, or clearing (sending zeros) its sleep value.

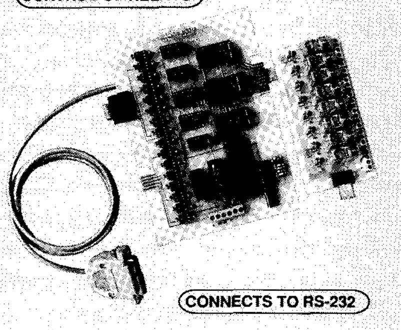
Internally, the real-time executive does not actually check all of the timers, tasks, and time slices at each clock tick. Doing so would be far too time consuming. Instead, whenever any time value is changed, that is, whenever a task is scheduled to run, either sleepTask(), sleepTill(), setTimer(), or setTimer_id() is called. Then the real-time executive computes the value of the next nearest clock tick time and

Listing 4—*This sample event function returns true (nonzero) when keyboard data is available or* false *(zero) when no keyboard* data *is available.*

```
;''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''';
;  KeyboardEventFct                                          ;
;.................................................................................;

        entry   keyboardEventFct        : emulates Pascal conventions
                                        ; <must be in rev stack order>
        darg    -argument               ; void far * argument
        arg     __event_id              ; event event-id

        push    bx
        push    ds
        mov     ax,40h
        mov     ds,ax                   ; look at bios kbd area
        mov     bx,1Ah

        cli
        mov     ax,word ptr [bx]        ; 1Ah
        cmp     ax,word ptr [bx+2]      ; 1Ch
        sti

        mov     ax.0                    ; false if no input available
        jz      keyboardEventFct_08     ; if zero, no keys pending ->

        mov     ax.1                    ; true if input available

keyboardEventFct_08:
        pop     ds
        pop     bx
        return  pascal

keyboardEventFct endp
```

RELAY INTERFACE

PROVIDES SOFTWARE CONTROL OF RELAYS

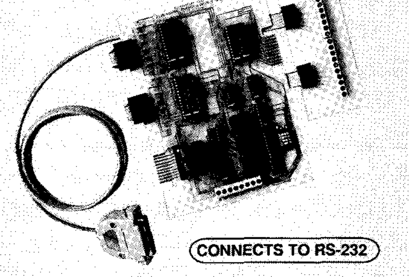


CONNECTS TO RS-232

AR-16 RELAY INTERFACE........................$ 89.95
Two 8 channel relay output ports are provided for control of up to 16 relays (expandable to 128 relays using EX-16 expansion cards). Each relay output port connects to a relay card or terminal block. A variety of relay cards and relays are stocked. Call for more info. RS-422 available (distances to 4,000 feet). PS-4 port selector may be used to control satellite AR-16 interfaces. (up to 16,384 relays)
RD-8 REED RELAY CARD (8 relays, 10 VA).........$ 49.95
RH-8 RELAY CARD (10 amp SPDT 277 VAC).......$ 69.95
EX-16 RELAY EXPANSION CARD (16 channel)...$ 59.95

ANALOG TO DIGITAL

8,10 & 12 BIT RESOLUTION



CONNECTS TO RS-232

ADC-16 A/D CONVERTER **(16** channel, **8 bit)..$** 99.95
Input temperature., voltage, amperage, pressure,energy usage, energy demaand,light levels, joystick movement and a Wide variety of other types of analog signals. Inputs may be expanded to 32 analog or 128 status inputs using the AD-I6 or ST-32 expansion cards. 112 relays may be controlled using W-16 expansion cards. Analog inputs may be configured for temperature input uinq the TE-8 temperature input conversion. RS-422 available. PS-4 port selector may be used to connect satellite ADC-16 interfaces (up to 4,096 analog inputs/16,384 status inputs and 14.336 relays). Call for info on 10 & 12 bit converters. [terminal block and cable sold separately)
ST.32 STATUS EXPANSION CARD...................... **$ 79.95**
Input on/off status of relays, stitches. HVAC equipment, thermostats, security devices, smoke detectors and other devices including keypads and binary coded outputs. Provides 32 status inputs (opto isolators sold separately),
TE-8 TEMPERATURE INPUT **CONVERSION........** $49.95
ncludes 8 temperature sensors & terminal block. Temperature range is minus 40 to 145 degrees F.
PS-4 PORT SELECTOR (4 channels RS-422).......$ 79.95
Converts an RS-232 port innto 4 selectable RS-422 ports.
TOUCH TON**E DECODER** and other serial interfacing products available. Call for tree information packet.

- FULL TECHNICAL **SUPPORT**...Provided over the telephone by our staff. EACH ORDER INCLUDES A FREE DISK WITH PROGRAMMING EXAMPLES IN BASIC, C AND ASSEMBLY LANGUAGE. A detailed technical reference manual is also included.
- **HIGH RELIABILITY**...engineered for continuous 24 hour industrial applications. All ICs socketed.
- Use with IBM and compatibles, Tandy, Apple, Mac and most other computers with RS-232 or RS-422 ports. All standard baud rates and protocols may be used (50 to 19.200 baud).

Use our800 number to order FREE INFORMATION PACKET. Technical Information (614) 464-4470.

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

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

ELECTRONIC ENERGY CONTROL, INC.
380 South Fifth Street, Suite 604
Columbus, Ohio 43215

#111

passes it **along to the real-time clock** ISR.

When the clock timer interrupts and detects that the time of any one of the timers has elapsed, the evaluation of the timers and the scheduler is called. The call is made to any expired timer function, sleep tasks are reactivated as expected, and the result is the execution of the timed function. Once the timer function has been executed, all of the timers and task sleep values are again checked, and the next nearest clock tick time is computed and passed to the real-time clock ISR.

INTERRUPT SERVICE ROUTINES

ISRs must inform the real-time system when hardware-detected events have occurred. All ISRs must call the scheduler in order to determine which event causes which task to run. Obviously, an interrupt that did not detect any event need not call the scheduler.

Within the PC environment, letting the real-time system know that an event has occurred and calling the scheduler just prior to the IRET instruction within the ISR are all that are really necessary. Whether you call the scheduler before or after restoring the registers saved by the ISR does not matter, nor does whether interrupts are enabled or disabled. Listing 3 shows how the ISR calls the scheduler.

The real-time system knows the event has occurred only if the flag is set in the SystemEvents array or if there is an event function for the pertinent events supported by this ISR.

To set the bit flags in the SystemEvents **array, use the** setSystemEvent() function. The event flag will be set. There are clearSystemEvent() **and** toggleSystemEvent() functions as well.

An event function is particularly well suited for ISRs. You need to write a function that returns a True or False status for the event ID passed to the routine. For example, an event function for the keyboard status would test characters available in the BIOS save area and return a True (a nonzero) or a False (zero) status.

Listing 4 shows a sample event function for the keyboard.

THE SCHEDULER

At this point, I have explained almost all of the real-time system except for the scheduler, the central part of every real-time executive that determines the next task to run. The scheduler is called from a number of circumstances and performs the same action in all cases, making its job straightforward.

The scheduler may be called by an ISR when the it detects an event. The ISR sets the SystemEvents (using the setSystemEvent() function), and then calls the scheduler, which determines whether or not to run another task.

The waitEvent() and suspendTask() functions call the scheduler. The current task needs to wait for an event. Again, the scheduler determines whether or not to run another task.

When the scheduler is called, it saves the current machine state, including all of the registers, in the current task's stack. The task's restart address is already saved on the stack by the scheduler function call. Finally, the current value of the stack pointer is saved in the task's entry in the *tasks* array. At this point, the current task can be suspended because all relevant and important information has been saved.

The scheduler determines which task to run next by testing, in order of priority, each task's ability to run. If a task is not sleeping, then the scheduler determines if there are any waiting events. If a task has no waiting events in its task entry, then it is ready to run. Otherwise, the task waiting events are matched against the SystemEvents. Amatchwillpermit the task to be reactivated, and the reactivated task may be the current task if there are no other tasks waiting to run.

When a task is ready to be reactivated, the stack address is pulled from the task's entry in the *tasks* array, and all of the registers are restored. A return is made back to the caller and is where the task activated by the

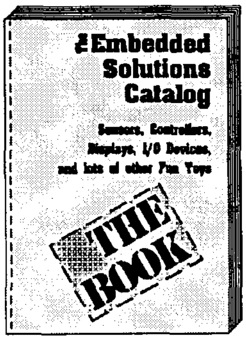
Looking for temp probes, pressure sensors, A/Ds, LCDs, enclosures?

Then **Get The Book!**

Why **waste** time chasing components and controllers? Now there's a single, easy source for thermocouples, thermocouple interfaces, barometric sensors, position sensors, rotary encoders, LCD displays, enclosures, and more. Use these with *our* **nanoLINK** controllers or with your own.

Free!

- Competitive prices on quantities as low as 1.
- Informative run-downs on each component--not just product listings and technical mumbo-jumbo.
- Every item supplied with technical information and application notes.
- New! Complete line **of nanoLINK peripherals and accessories.**


Embedded Solutions Catalog
Sensors, Controllers, Displays, I/O Devices, and lots of other Fun Toys
THE BOOK

To get your free copy of The Book,
Call 1-800-GET-DATA
or fax l-602-996-0255

nano Power Devices
A division of AirDigital Corporation

#1'

scheduler was last suspended. Thus, the task is completely restarted.

MS-DOS

MS-DOS was written to support a single-user, single-application program. It expects to continue executing a command to completion. In order to coexist successfully with DOS, the real-time executive has its own internal protection mechanisms.

Normally, outside of the real-time executive, hardware interrupts present no problem for MS-DOS. When an interrupt occurs, registers are saved, then restored, and DOS continues.

With the real-time executive, a hardware interrupt may cause the scheduler to run. If the scheduler runs, another task may be started. Not only does DOS have no chance to complete as expected, but another task may actually execute DOS commands leading to serious confusion.

A switch implemented by the real-time system called insideDos is incremented whenever a DOS command is executed by any code inside and outside of the real-time executive and decremented when the command completes. You trap all DOS calls by building a shell around DOS. All tasks make normal calls to DOS, which are redirected to this shell.

When a hardware interrupt leads to a scheduler call, the scheduler checks the insideDos flag. If it is nonzero, the scheduler knows that the interrupt occurred while in DOS and does not run. The scheduler sets its own schedulerPending flag, and the DOS command is allowed to complete.

A check is made for the schedulerPending flag when the DOS command completes and returns to the DOS shell you have built. The task, now outside DOS, can be suspended and the scheduler allowed to run.

THE DEMO PROGRAM

The demo program divides the screen into four squares. In the first square (top left], the demo program shows the priorities of each task. This task waits on keyboard events. It processes cursor keys to adjust the priority of each task.

The time slice allocated to each task can also be changed by making the time-slice interval slower [press S) or faster (press F). The time slice can be restored to normal by pressing N.

In the next two squares, you should see text files scrolling as fast as possible. This aspect is handled by two independent tasks. Neither task actually waits on any event, instead they are time-slice interrupted, giving each task a chance to run. The default time slice is defined in the header files as half a second.

The last square displays the time of day. A timer function is set to call code for this display.

BUILDING YOUR OWN APPLICATIONS

Now it's your turn to use the real-time executive to build your own applications. Included with the source code is a custom. h file to quickly tailor some of the major parameters used by the real-time executive.

These parameters include the maximum number of tasks, maximum number of timers, the minimum stack size to allocate, the maximum time slice for a task, and the all important custom event names. You may want to add events to the system events section of the main header file.

Just add your own tasks and stir! ❑

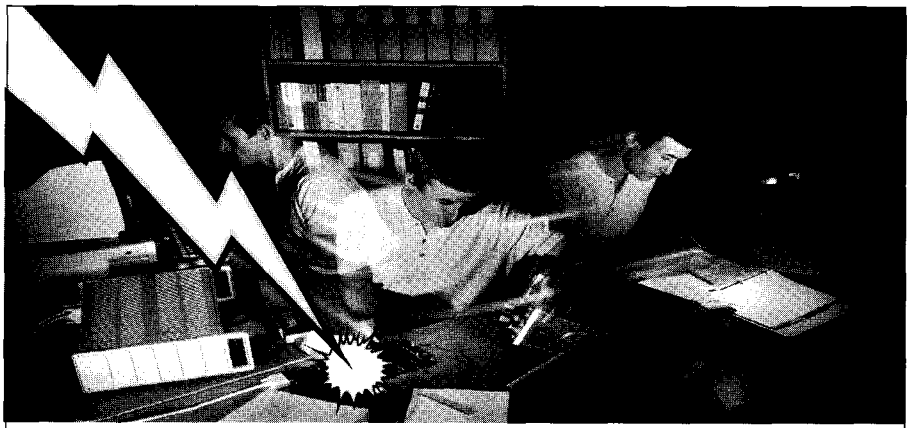
Michael Podanoffsky has spent the last 20 *years as a software developer building real-time systems, multiuser networked databases, and* language *compilers.*

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

401 Very Useful
402 Moderately Useful
403 Not Useful



MultiTask!™ Execs and GOFAST™ Math Speed Time-to-Market

Zap your application with peak performance using tested code and expert support. Control real-time scheduling with MultiTask! **source code** executives, or **ROM-able, re-entrant** GOFAST floating point:

*Replace 80x87 MATH **COPROCESSORS**;
*Drop-in IEEE SOURCE LIBRARIES;
*Link-and-Go with C compilers: Intel®, Microsoft®, Borland®, WATCOM®, Zortech, Metaware®, and more..

Solutions for **80386/486 PROTECTED-**MODE, 80x86 & V-Series, Z80/180/64180, 8085, 68xxx, **68HC16**, 68HC11, 6801, 6809, 805 1, 80 196, i960, R3000, SPARC® and more.

Call for **free** information diskettes today. PHONE 503-641-8446; FAX 503-644-2413; USA TOLL FREE **800-356-7097**.

14215 NW Science Park Drive
Portland, OR 97229
U S SOFTWARE®

©1992 US software Corporation GOFAST and MultiTask! are trademarks of US Software Corporation All other trademarks belong to their repective owners

#113

FEATURE ARTICLE

Robert Scott

Resource Management in Cooperative Multitasking

In applications where cooperation between tasks is necessary to get the job done, sharing hardware resources effectively can make the difference between a useful system and a useless system.

Cooperative multitasking has been defined in literature and compared with other types of multitasking, such as time-slicing. In cooperative multitasking, the tasks themselves decide when to give up control to the scheduler, passing control to the next task. The problem of resource sharing in multitasking has been studied in various textbooks [1]. Therefore, I will review the characteristics of cooperative multitasking as they relate to the effective management of resources.

You can implement cooperative multitasking with a very simple scheduler. All the scheduler has to do is keep track of the stacks and certain registers for each task. These stacks are used primarily by the task programs, but the same stacks can also be used by the scheduler to store registers when a task gives up control. The tasks are all assumed to be part of a single application. This scenario is very different from one in which a task represents a user on a time-sharing system. Instead, it is more like the case of an industrial processing system where each task represents a distinct concurrent function.

For example, an 80x86-based cooperative-multitasking scheduler managing programs written in Microsoft C using the small memory model need only save the SI, DI, and BP registers. The CS, DS, and SS registers are common in this model and do not need to be saved. Because all floating point operations are completed between calls to the scheduler, there is no floating point coprocessor information to be saved. When multitasking is running under MS-DOS, a nonreentrant operating system, problems are avoided because all calls to MS-DOS are completed before a task gives up control. If cooperative multitasking is implemented on a 680x0-based system, then the entire register set needs to be saved by the scheduler because any of the address or data registers could be used as register variables.

The only resources the scheduler itself manages in cooperative multitasking are the stacks and the order of task execution. The responsibility of managing all other computer resources is left up to the tasks. In many cases, the application requires little in the way of resource management; however, one resource that all applications must manage is CPU time. Each task must deliberately give up control to the scheduler within a reasonable time. What is reasonable depends on the application. For example, if one of the tasks is a closed-loop speed control for a DC motor requiring a feedback response time of less than 100 ms, then that task must get control at least once every 100 ms. Without a time-slicer in charge, you must ensure all tasks cooperate by having them give up control fast enough to satisfy the timing requirements of the closed-loop task.

In general, cooperative multitasking does not afford very fast guaranteed response time without imposing very strict requirements on the program structure. But if the application naturally has frequent occasion to wait for external events, then during those pauses the task can give up control to the scheduler.

AN EXAMPLE

In my work with software that is controlling automated test equipment (industrial test stands), I often need to control several identical but independent test stations with the same computer. Each test station runs a full set of tests on some manufactured part. Using cooperative multitasking allows one copy of the test program to control all the test stations. Each instance of the test program is a task in the system.

Let me use this example to illustrate the various aspects of resource management in cooperative multitasking. Note that this example is special in one respect: several tasks run the same code. In the more general case, each task is a different program. With one block of code run by several tasks, occasionally the need for task differentiation arises. For instance, differentiating would be necessary when deciding which physical output needs to be set when the code calls the function clamp_part(). If there are four testing stations and four clamp actuators, the function clamp-part() must know which clamp to actuate.

When you need to differentiate tasks, a system variable (maintained by the scheduler but accessible to the tasks) is used to identify which task is currently in control. Then primitive functions like clamp_part() can reference the task identifier to determine the actual physical I/O involved.

Listing 1-Dynamic *response-time CPU sharing relies on knowing the desired response lime and the expected execution time of each task.*

```
alloc_time(nr,ne) {
   int sum,i,j;

   if  ((nr<r[tasknum])|| (ne>e[tasknum])){
      do {
         for  (j=0; j<=taskmax; j++) {
            sum = 0:
            for  (i=0; i<=taskmax; i++)
               if (i!=j)
                  sum += e[i];
            if (r[j] < sum) {
               suspend();
               break:



      }
      while (j<=taskmax);
   }
   r[tasknum] = nr;
   e[tasknum] = ne:
```

SHARING MEMORY

If you allocate temporary variables in registers or from the stack, then memory allocation is automatic, and no special precautions need to be taken for multitasking. But watch out for static variables! If two tasks operate on the same static variable, they will interfere with each other unless that variable is specifically for intertask communication. If a static variable is not for intertask communication, then the variable must become an array, and all task accesses are then indexed by the task number.

CUTTING UP DISPLAY SCREENS

To share a display screen between tasks, you as the programmer can divide up the screen (horizontally or vertically) so each portion of the screen is owned by a single task. For example, a four-station tester can divide up a 24-row screen giving each task a band of six rows. But even if different tasks use different portions of the screen, there is still a shared resource that cannot be divided: the cursor. There is only one cursor and it can only be in one place at a time.

Making all uses of the display screen begin with a cursor-positioning command and completing the output to the screen without any intervening calls to the scheduler is one way to manage the cursor as a shared resource. This approach is the simplest one, but it also has the worst response time, particularly if the display screen is on a slow serial interface, or if large blocks of data must be displayed. This delay exists because a task retains control for the entire time it takes to write its message to the screen.

Response time is better if the display screen is memory mapped, or in the case of the serial interface, if output to the serial port is interrupt driven through a large enough buffer so the buffer never fills up. Another alternative is to limit the message length by dividing up long messages into several short ones, relinquishing control between each short message.

All these methods assume each task wants to write to the display. But often having a single task [a ***monitor task)*** do all the writing is better. For example, a monitor task would be necessary if there had to be some concurrent user interface that operated independently of the testing tasks. The monitor task could give up control between each character sent to the screen, and if any task wants to write to the screen, it would have to pass the message through the monitor task.

SHARED I/O HARDWARE

A data acquisition board is a common shared resource in the field of industrial test stands. For instance, a 16-channel ADC may be shared by several test stations. If each test station owns a separate set of these analog inputs, then device sharing is easy as long as each use of the device is uninterrupted by calls to the scheduler. With fast A/D conversion times, a task can afford to trigger a conversion and wait for the results all on one scheduler turn. But if the ADC board is too slow, special considerations are needed.

When a task wants to use the ADC board for any period of time longer than one scheduler turn, it can check and then set a global *use flag* (if it was previously clear) or wait otherwise. The use flag locks out all other tasks from using the board until the task that set the flag clears it.

The use flag is a special case of an operating system mechanism known

AMX™

REAL-TIME
MULTITASKING
KERNEL

3086/88, 80x86/88 80386
Z80, 64 180, 8080/85 68000/10/20

- Fast, reliable operation
- Compact and ROMable
- PC peripheral support
- DOS file access
- C language support
- Preemptive scheduler
- Time slicing available
- Configuration Builder
- Complete documentation
- Intertask messages
- Message exchanges
- Dynamic operations
  - — task create/delete
  - — task priorities
  - — memory allocation
- Event Manager
- Semaphore Manager
- List Manager
- Insight'" Debugging Tool

THE BEST
Join over 1000 developers such as
IBM@, Xerox, Hewlett Packard,
Hayes, Hughes Aircraft and NASA.

CHOOSE AMX
The best low-cost, high-performance
real-time multitasking system
available today.

No Royalties
Source Coda Included

Demo Disk and
Manual only $85 US Call for prices for
AMX 86 $3000 US other processors.
(Shipping/handling extra)

IBM is a registered trademark of IBM Corp
Z80 is a trademark of Zilog,Inc
AMX. AMX 86, InSight are trademarks of
KAOAK Products Ltd.

KADAK Products Ltd.
206-1847 West Broadway
Vancouver. B.C.. Canada
V6J 1Y5
Telephone: (604) 734-2796
Fax: (604) 734-8114

#

as a *semaphore.* Time-slicing systems require special precautions when implementing because other tasks may modify the flag after it has been checked by the first task. But in cooperative multitasking, each block of code between calls to the scheduler is automatically a *critical region.* It cannot be interrupted by the scheduler without the cooperation of the task. Therefore, semaphores and other global variables change in a more predictable manner.

Just to show how elaborate resource sharing can become, consider the following application. An ADC board is programmed to trigger periodic conversions based on a timer and cause an interrupt when a conversion is complete. This programming is necessary in the application because some logical decisions need to be made based on the results and the required sample rate is too high to be ensured by cooperative multitasking.

After each conversion, the interrupt service routine takes its instructions for the next conversion from a global table of channel numbers. Each task in the system is allocated one slot in the table, and all communications with the ADC board are carried out through this table and the interrupt service routine, which is not considered part of any one task.

In addition to the dedicated table entries for each task, there is also an extra table entry that may be allocated to any task by means of a use flag. This way, a task can have fast access to its dedicated table entry and slower access to the shared entry for less time-critical readings.

HOW MUCH TIME DO YOU NEED?

I mentioned CPU time as the most obvious resource that needs sharing. But instead of imposing one uniform response-time requirement for an application, varying the response time during the running of the application is often advantageous.

Returning to my example of a multistation tester, suppose each station has a test sequence fairly noncritical as to response time, but has a short section of the sequence where faster response is needed. Suppose a hydraulic solenoid valve is actuated and the response time of the resulting oil pressure needs to be measured to within 2 ms. The straightforward way to ensure fast response time is to have the task retain control by not calling the scheduler for the duration of the test (from the time the solenoid is actuated to the time the oil pressure responds). This arrangement suspends multitasking for what could be an indefinite period of time, which will probably have unacceptable effects on the other tasks.

TEMPORARY MONOPOLY

You can mitigate the effects somewhat by having the task request permission to retain control from the other tasks through a semaphore. This way, the requesting task will not begin the time-critical portion of its test sequence until it has permission to complete that portion of the sequence with multitasking disabled. Also, the other tasks will only be interrupted during a period of time in which the indefinite suspension of multitasking is less serious. If most of the total test time is taken up by the wait for an initial condition to be met (like flooding the part with oil), then occasionally suspending multitasking would only affect testing throughput, and only slightly at that.

But this aspect brings up another problem. What if the other tasks never give permission for the requesting task to run all by itself for a while? The requesting task could lock up indefinitely waiting for this permission. There are clearly special application-dependent conditions that must be met. This method of allocating an extended CPU monopoly works under the following conditions:

1. when other tasks give permission for a CPU monopoly frequently enough that the requesting task is guaranteed to be granted permission within an acceptable length of time

2. when a task is requesting permission to monopolize the CPU, which also gives permission for other tasks to do so (no deadlocks)

3. when a task gives permission for other tasks to monopolize the CPU, which allows it to tolerate all of

the other tasks taking advantage of that permission for the maximum time allowed for such monopoly

4. when the overall effect of granting the occasional extended monopoly to a task has an acceptably small effect on the system throughput

How likely are such conditions met? That depends on the application. If the maximum length of time that a task monopolizes the CPU is small, and requests for such monopolies are infrequent, then this method will likely work. If not, then more sophisticated methods are needed.

A GENERALIZATION

The CPU needs of a task may be characterized by two dynamic parameters: the response time that the task needs from the scheduler and the response time the task is willing to give to the scheduler.

The first parameter refers to the maximum period between the time the task gives up control and the time control is returned to it by the scheduler. For task *i,* call this response-time parameter r_i. The second parameter refers to the maximum period during which the task will retain contrrl before calling the scheduler. For task *i,* call this execution-time parameter e_i. At any moment in time, the CPU time is said to be acceptably allocated for all tasks if the following holds:

For each task *j,* $r_j > \sum_{i \neq j} e_i$ (1)

This inequality assumes task-switching time is either negligible or is figured into the e_i values. The inequality merely states that the response time required by a task allows for the worst-case execution times of all the other tasks.

In the monopoly solution, inequality (1) is satisfied by having two constant response-time requirements for each task. The smaller one, r_{fast}, is the response time required by a task when it is not giving permission for monopoly by other tasks. The larger response-time requirement, r_{slow}, is the response time required by a task when it is giving permission for CPU monopoly. Similarly, the usual execution time for a task is e_{fast}, but when the task is engaged in CPU monopoly, its maximum execution time becomes e_{slow}. Inequality **(1)** is satisfied for n tasks if both r_{fast} and r_{slow} are at least n-l times as large as their respective e_{fast} and e_{slow} counterparts. But the conditions may be improved to

$$r_{fast} > (n-1) \times e_{fast}$$

and

$$r_{slow} > (n-2) \times e_{fast} + e_{slow}$$

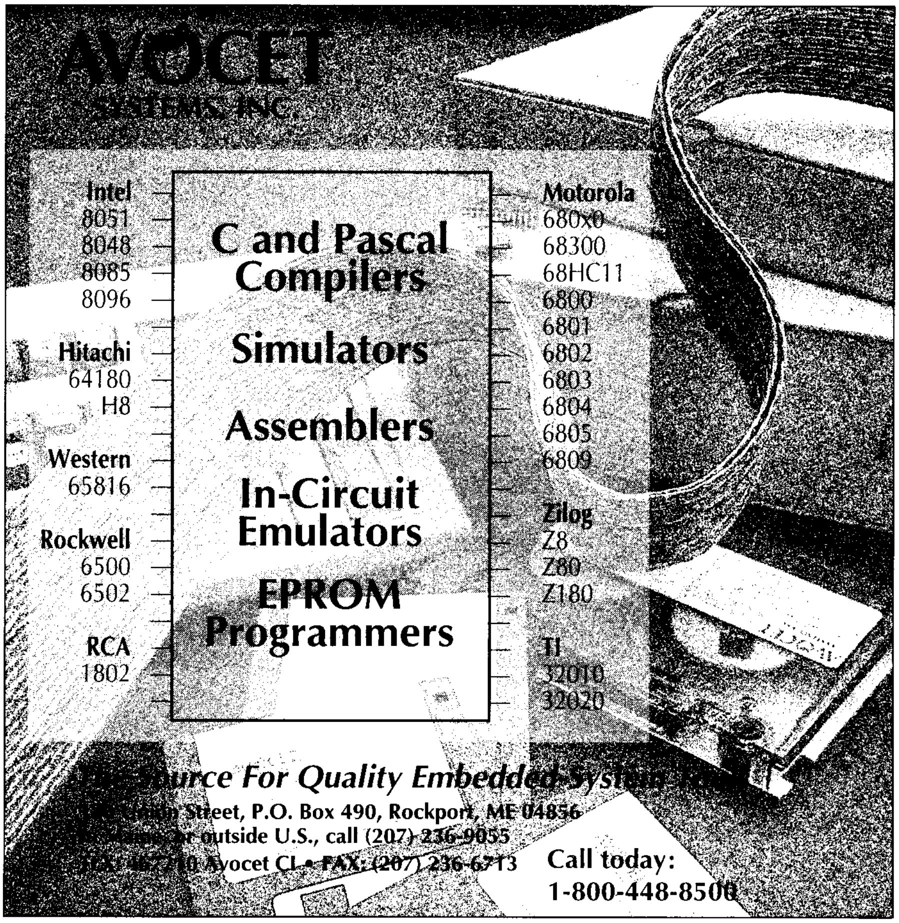
by simply requiring that, after one task has exercised monopoly, no other task may do so until one complete round of the scheduler has occurred. That way only one monopoly needs to be allowed for by the nonmonopolizing tasks.

These observations lead me to the question: "Can CPU time sharing in cooperative multitasking be refined by a more precise control of the execution-time and response-time parameters?" There does seem to be room for improvement.

In the general case, each task can dynamically maintain its own r_i and e_i. Certain rules can be established by which tasks manipulate these parameters. A task may freely raise its response-time requirement or lower its execution-time limit at any time, but movement in the other direction must be coordinated with the other tasks.

For example, if task **#1** has started a time-critical sequence based on the current maximum execution times of other tasks, then other tasks may not arbitrarily raise their execution times. If they did, then task **#1** would be placed in the awkward position of having to abort a time-critical sequence to which it had already committed itself. Similarly, a task may not arbitrarily lower its response-time requirement unless such a lowering is consistent with the execution-time requirements of all the other tasks (inequality (1)). This aspect leads to two criteria for changing r_i and e_i parameters:

1. Task *i* may raise e_i only if the new value of e_i satisfies (1) for all other tasks, $j \neq i$.



#116

2. Task i may lower r_i only if the new value of r_i satisfies (1) for the task

In the special case of temporary monopoly I described earlier, giving and revoking permission for monopoly is equivalent to changing between the r_{fast}/e_{fast} parameters and the r_{slow}/e_{slow} parameters. In that case, the two change criteria are automatically satisfied because when a task takes advantage of permission to hog the CPU, the other tasks that gave permission don't even get a chance to run until the monopolizing task is done. Therefore, those other tasks never get a chance to revoke their permission until the use of that permission has been completed.

AVOIDING DEADLOCK

If the two criteria listed above are used by all tasks, then you are guaranteed the required response time for each task will always be satisfied. The assumption is if these criteria prevent a task from adjusting its r_i or e_i parameter, then the task must keep waiting until such an action becomes allowable. In general, there is no guarantee this will happen in any reasonable length of time. Several tasks could set their parameters so no task can advance to its next required r_i and e_i parameter values.

In the special case of CPU monopoly discussed earlier, condition #2 for that method prevents deadlocks. That is because whenever a task is waiting for more resources (a lower r_i or a higher e_i), it is claiming the least amount of resources. A similar rule can also be established in the general case. Whenever a task wants to increase its resources, it must first declare ownership of minimal resources.

IMPLEMENTATION

Dynamic response-time CPU sharing may be implemented completely within task programming; it does not have to be part of the scheduler. You can make a global array of response times and execution times available to all tasks. The units of time referred to in the r[] and e[] arrays may be arbitrary. No actual real-time measurement of these times is implied. However, you as the programmer still have to calculate execution and response times in some form, but the cooperative multitasking scheduler does not have this problem.

Listing 1 shows a C function you may use to implement CPU sharing. Assume that the global variable tasknum **is the task number for the currently running task, and** taskmax is the maximum task number. The function alloc_time() takes two parameters, nr and ne. Parameter nr is the desired new response time and parameter ne is the desired new execution time. The function alloc_time() **does not return to the** task until it succeeds in changing to the new parameters. The function suspend() is the call to the scheduler that gives up control.

Although alloc_time() will handle the general case, it should not be necessary in most applications. Having a set of programmer-defined states (such as the two-state monopoly model) in which response and execution-time requirements have been checked by hand is a much more efficient system to run.

CONCLUSION

While offering a very simple framework in which to schedule tasks, cooperative multitasking pushes more of the problem of satisfactory resource management onto the task programs themselves. If these problems can be solved in a way not too cumbersome to the application, then the simplicity of the scheduler may be preserved. ❑

Robert Scott is the owner and chief engineer of Real-Time Specialties, a developer of custom software for embedded systems.

REFERENCE

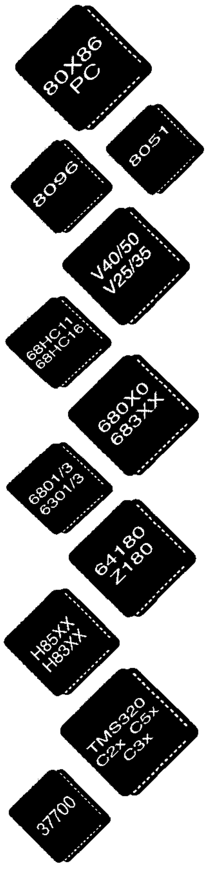
[1] P. B. Hansen, ***Operating System Principles,*** Prentice Hall, 1973

I R S

404 Very Useful

405 Moderately Useful

406 Not Useful



Get ***Byte-BOS***™ and leave the MULTITASKING to us!

Byte-BOS Multitasking Operating System (Byte-BOS) is designed for use in real-time applications on a PC and a wide range of embedded processors. After several years of proven performance in the field, Byte-BOS is now available for **only** $495 for **PCs** and **only** $995 for embedded processors.

Byte-BOS is complete, requires no "add on" components, and includes **all** theses features:

- □ preemptive task scheduling
- □ **timeslicing**
- □ **multithreading**
- □ dynamic & static. task management
- □ multiple task events
- □ multiple task message buffers
- □ multiple task timeouts
- □ global resource management
- □ **fixed** block memory management
- □ on **chip & PC** serial I/O management
- □ external serial I/O management
- □ **on** chip & PC timer management
- □ nested ISR & ISR stack management
- □ low power state management
- □ non-blocking & timeout **return** modes
- □ **configured** to **your** c compiler
- □ working application for **popular** targets
- □ no royalty ANSI c **source** code
- □ 250 page reference manual
- ❑ 1yearoftechsupport&updates
- □ **BOSVIEW™ multitasking** monitor
- □ **profiler** for task **& code** section **timing**

Call today to order, or request a detailed **information** packet. With the time and money you save, **you'll** be glad you left **the** multitasking to Byte-BOS.

Byte-BOS™ **Integrated Systems**
PO Box 3067 Del Mar CA 92014

800-788-7288 or 619-755-8836

#117

FEATURE ARTICLE

Francis Lyn

Using Programmable Logic Controllers

We hear a lot about general-purpose microcontrollers used in embedded control situations, but what about the die-hard controllers that brave the hostile environment of the factory floor? Find out about PLCs.

The application of Programmable Logic Controllers (PLCs) is generally limited to the small segment of the electrical community involved in industrial control applications. I'd like to introduce you to the exciting world of PLCs. I will do so by describing the design of an improved low-cost PLC engine that runs on a popular microcontroller platform.

Since its invention in the early 1970s by Dick Morley, the PLC has evolved into a universally accepted building block used in most logic control systems in industry today. The PLC is available in sizes ranging from plant-wide systems using several large PLCs (each supporting thousands of I/O points) connected together by a local area network and costing millions of dollars, to a single small PLC with several I/O points and a hand-held programmer costing a few hundred dollars.

Basically, a PLC is a special-purpose computer designed to replace hard-wired relay logic circuits and built to survive in the electrically and mechanically harsh environments of typical industrial applications.

A PLC makes use of a computer's processing power to provide more than just simple relay logic replacement functions. Delay timers, real-time clocks, flip-flops, edge-triggered inputs, barrel shifters, event counters, communication functions, math functions, and user-defined macros are only some of the advanced functions a modern PLC provides.

I/O PROCESSING

A PLC may be viewed as a black box with a number of input and output terminals for connecting it into an electrical circuit. The I/O signals are digital; they are either ON or OFF just like the signals in a traditional relay control circuit. Inside the black box, the PLC processor runs several real-time tasks, such as the execution of the user's program, servicing communications requests, maintaining real-time timers, and servicing the I/O by automatically reading and writing to inputs and outputs on a regular basis.

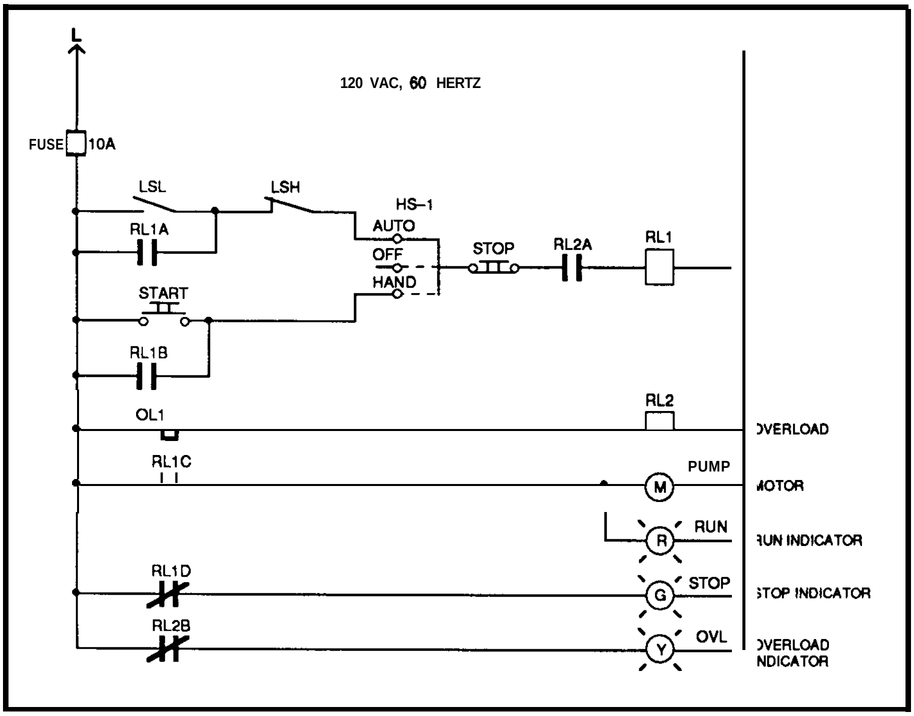
The PLC maintains a special buffer area called an I/O table, where it stores a bit image of all the input and output conditions. When inputs are read, the input bits in the I/O table are set or cleared to make a snap-shot image of the input signal conditions. When outputs are updated, the I/O table output bits are written to the output terminals. I/O servicing is done automatically for all I/O points, even when not all are used by the application program. Details of the servicing task are hidden and the user may ignore them.

The user can determine the state of the inputs and can control the state of the outputs by reading from and writing to the I/O table. The bits in the table represent the only "data" the processing unit operates on, so they are all the application program needs to know about.

THE APPLICATION PROGRAM

The user's application program is simply a list of instructions that directs the PLC on the sequence of tasks to be performed. The program is repeatedly executed during normal PLC operation, and one complete pass through the program is called a *scan cycle.*

The application program specifies the logic equations that describe the



Fiiun 1—A sample PLC application might be controlling an electric waler pump.

control scheme the PLC is implementing. As the program steps execute, the processing unit substitutes the input status value stored in the I/O table into the logic variable in the expression being solved, then it stores the resulting output state in an output bit of the I/O table.

TYING INTO THE REAL WORLD

A PLC would normally be mounted close to the equipment it is controlling, either in a control panel or in an electrical equipment room. The PLC I/O terminals are accessible on the PLC processor board or on add-on I/O modules connected to the PLC processor in the case of a modular system. The PLC is wired up to the circuit and to the devices that make up the control system; inputs are wired to monitor the signals from sensing devices and outputs are wired to indicating or control loads. The number of I/O terminals supported by the PLC may be fixed or expandable and is a basic sizing parameter for matching a PLC to an application.

PLCs usually operate in electrically harsh environments where inputs are likely to contain large amounts of noise-with the occasional voltage spikes thrown in-and outputs are normally called upon to drive loads with high in-rush currents or large reactive impedances. The typical PLC may be directly connected to the actual equipment it is controlling and may not have any buffering from the electrical conditions on the equipment. For this reason the I/O interface circuits must be designed and built to conservative standards.

Industrial PLCs offer a wide range of I/O options that cater to the range of interface voltage levels a PLC is likely to encounter. Voltages from 24 volts to 220 volts, AC or DC, are not uncommon. PLC input circuits may use transient suppressors, filters, and optocouplers or small relays for voltage isolation. PLC output options may include dry contact, triac, or transistor output devices. In special cases, using electromechanical buffer relays between the PLC and the control circuit may still be necessary; for example, one would be necessary to interface a PLC to a high-voltage circuit breaker.

BASIC PLC OPERATION

A basic PLC consists of a set of input terminals, a set of output terminals, and a processing unit. Input signals can come from any type of switching device, including mechanical switches, electromechanical and solid-state relay output contacts, hall-effect switches, optocoupler outputs, and so forth. Switch contacts are usually wired between a voltage source and a PLC input. The voltage source can be externally provided or may be supplied by the PLC system itself.

PLC outputs can be connected to any load device in the control system, such as indicating lamps, solenoid valves, motors, electromechanical and solid state power relays, contactors, and so forth.

The processing unit is the heart of the PLC. It handles such overhead chores as machine initialization, timer service functions, I/O servicing, communications functions, program loading, scanning and processing input signals, and executing the user's application program. Then, the unit updates the outputs by writing new results to them. Machine initialization is normally done once immediately after a power-up reset of the PLC.

I/O LABELS

Each I/O point supported by the PLC has a related bit in the I/O table. The application program may refer to each bit in the table by either a reference address or a label. Labels are usually easier to use than addresses because they can be assigned names that match the I/O terminals.

The application program fetches the status of an I/O signal through the reading of the labeled input bit and controls an output by writing the labeled output bit. The program also references internal bits in the same way by using their labels or addresses. Internal bits are data storage bits that hold intermediate results. These bits are named internal because they are not associated with I/O points.

A PRACTICAL EXAMPLE

Now I will show how a PLC may be used in a typical control application. The control elementary diagram of Figure 1 shows a small electric water pump controlled by manual start and stop push buttons and conventional relays RL1 and RL2. Indicator lamps are used for local status indication, and motor overload protection is provided.

The elementary diagram shows the electrical circuit and also defines the control logic by the circuit topology. Relay RL1 is energized by momentarily pressing the start PB. The RL1A contact seals in the circuit to keep RL1 energized when the start PB is released. RL1 can be deenergized by pressing the stop PB or by opening the RL2A overload contact. Under normal conditions, the motor thermal overload contact OL1 is closed, and RL2 is kept energized. On a thermal overload condition, OL1 opens, drops out RL2, and causes contact RL2A to open.

The first stage in any PLC implementation of a control circuit is to physically wire up the circuit by connecting all input devices to PLC inputs and all output loads to PLC outputs. Figure 2 shows a typical PLC wiring diagram for implementing the control circuit of Figure 1. Assume here that the PLC inputs are designed for 110 VAC voltage with a common return to neutral, and that PLC outputs are relay contacts with one side of each contact connected to a common terminal. Notice that the PLC wiring diagram shows only the wiring to the I/O devices connected to the PLC and does not convey any information on the required control logic. In fact, the order in which the input or output terminals are used does not matter as long as each device is wired to the PLC. The user's application program will define the transformation between inputs and outputs. The control scheme is described by the program, which must be developed and then stored in the PLC's program memory.

The next stage of program development is both an obstacle and a feature of the PLC solution to control problems. Writing PLC application programs requires some degree of knowledge and expertise, but the power and flexibility of the PLC approach far outweighs the initial time investment to learn how to program them. Because PLC designs vary significantly, the programming method is highly dependent on the chosen machine. In general, two popular types of programming methods exist.

LADDER AND FUNCTIONAL LOGIC DIAGRAMS

The ladder diagram method is an attempt to ease the programming effort by describing the program in terms of a ladder diagram, which closely resembles the elementary diagrams that electricians are used to. Ladder diagrams represent the physical wiring diagram of the control circuit where the logic expressions are embodied in the diagrams' circuit topology. For example, two switch contacts wired in series are equivalent to an AND logic function, or to an OR function if the switches are wired in parallel.

The other popular method of programming uses a functional diagram to describe the control logic scheme with logic function blocks [i.e., gates, flip-flops, macros, etc.). It is initially harder to learn, but it is far more powerful and easier to read and understand than the ladder diagram method. A functional diagram can precisely describe a logic statement without the distracting elements of the electrical circuit cluttering up the diagram.

The PLC program functional diagram in Figure 3 illustrates the use of function blocks to describe the control logic scheme. In comparison, note how much easier the functional diagram is to read and understand than the combined electrical schematic and control elementary diagram of Figure 1. With most well-designed PLCs, the user is able to write the application program directly from the functional diagram.

IDEAS FOR A PLC DESIGN

Now for a look at the operation of the typical modem small PLC and a suggested set of design specifications for improving the overall performance standards.

Most popular low-cost PLCs offer a hand-held programmer designed to operate only with that one type and model of PLC. To keep costs down, an LCD display and membrane-type keyboard are usually employed. A better approach

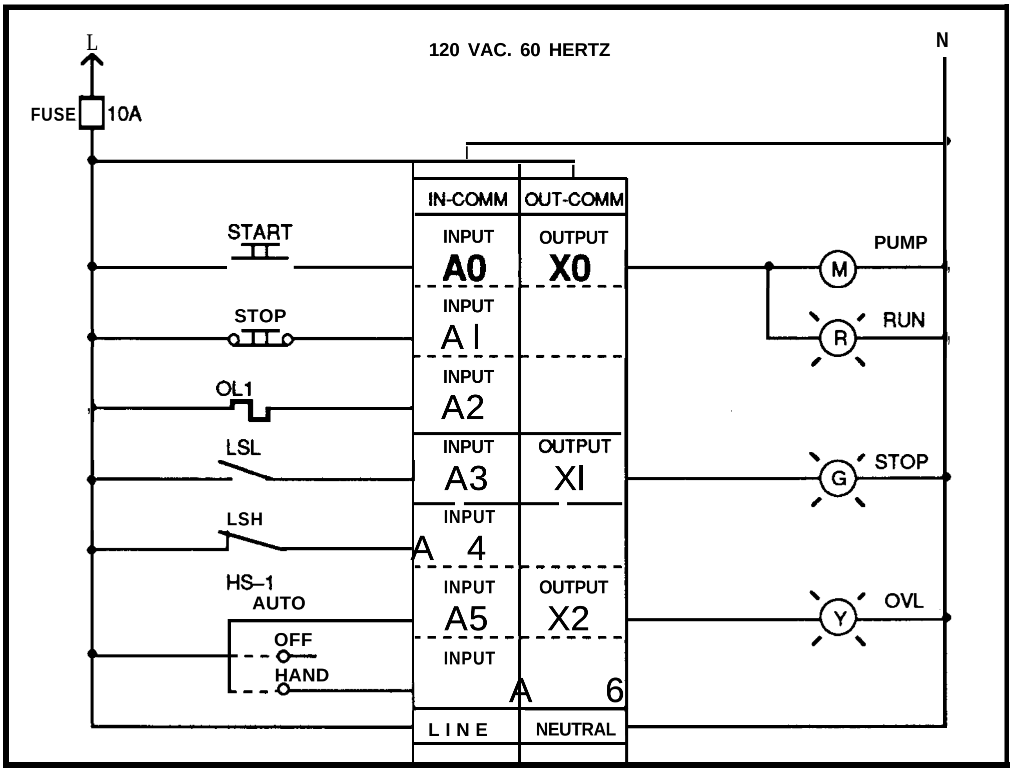


Figure 2-A *PLC wiring* diagram *for* the *circuit shown in* Figure 1.

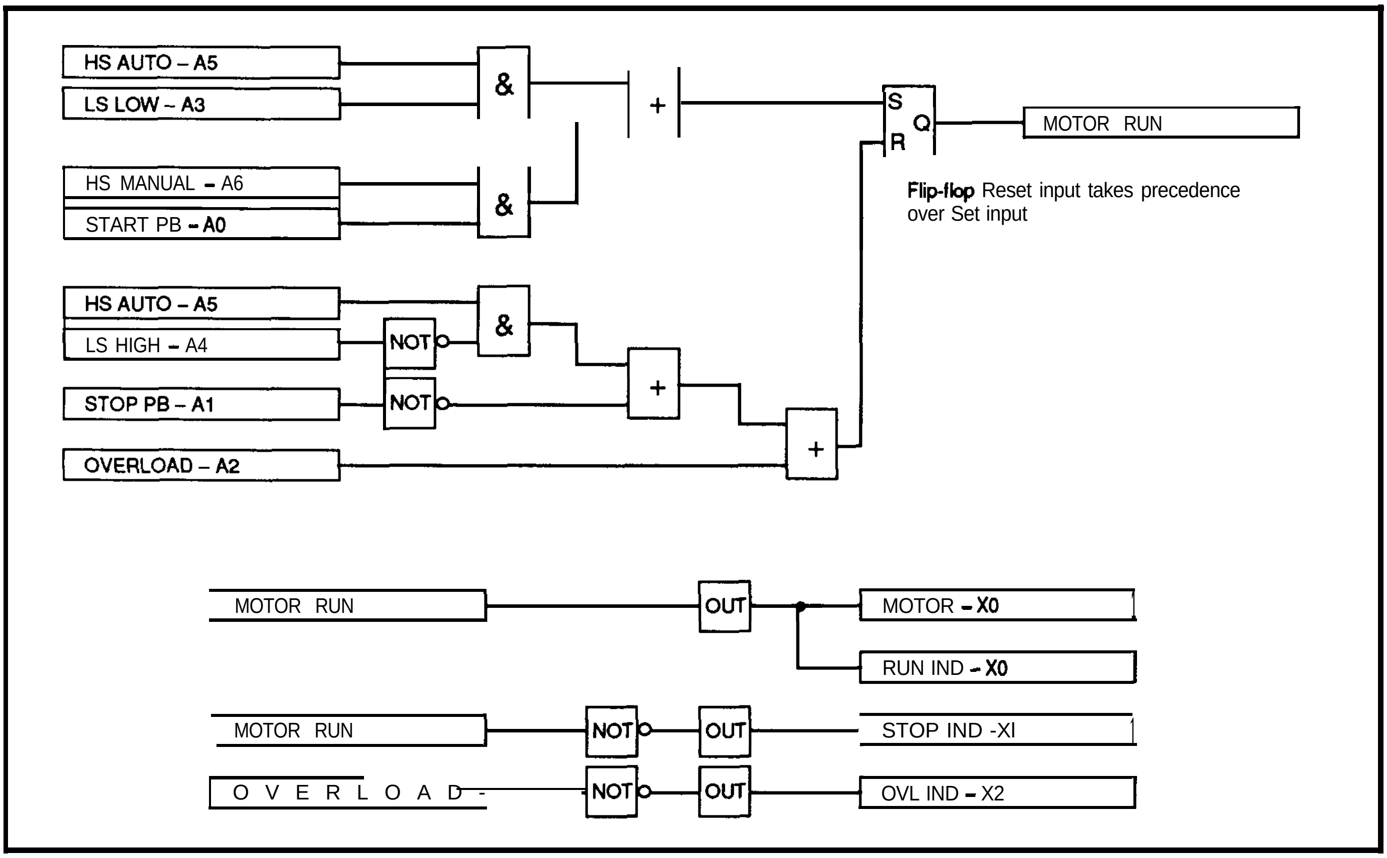


Figure 3—The PLC functional diagram illustrates the use of function blocks to describe the control logic scheme.

EPROM PROGRAMMERS

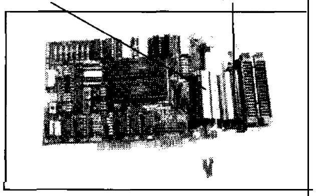
Stand-Alone Gang Programmer $750.00

• Completely stand-alone or PC-driven
*Programs E(E)PROMs
• 1 Megabit al DRAM
• User upgradable to 32 Megabit
. 3/.6" ZIF Sockets, RS-232, Parallel In and Out
• 32K internal Flash EEPROM for easy filmware upgrades
• Quick Pulse Algorithm (27256 in 5 sac, 1 Megabit in 17 sec.)
• 2 year warranty
. Made i" the U S A
*Technical support by phone
• Complete manual and schematic
• Single Socket Programmer also available. $550.00
• Split and Shuffle 16 8 32 bit
• 100 User Definable Macros, 10 User Definable Configurations
. Intelligent Identifier
. Binary, Intel Hex, and Motorola S
• 2716 to 4 Megabit

Internal Programmer for PC $139.95

New Intelligent Averaging Algorithm Programs 64A in 10 sec 256 in 1 min., 1 Meg (27010,011) in 2 min 45 sec 2 Meg (27C2001) in 5 min Internal card with external 40 pin ZIF

-Reads. Verifies, and programs 2716. 32. 32A. 64, 64A, 126, 128A 256, 512, 513. 010, 011, 301 27C2001, MCM 68764, 2532, 4 Megabits
• Automatically sets programming voltage
• Load and save buffer to disk
• Binary, Intel Hex, and Motorola S formats
• No personality modules required
• 1 Year warranty
• 10 days money back guarantee
. Adapters available for 8748, 49, 51, 751, 52. 55. TMS 7742, 27210, 57C1024, and memory cards
• Made in U.S.A.



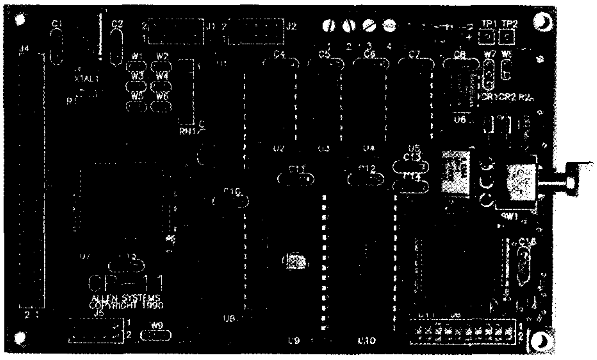
EMPDEMO.EXE available BBS (916) 972-8042

NEEDHAM'S ELECTRONICS
1539 Orange Grove Ave. -Sacramento, CA 95841
'Monday-Friday, 8 am-5 pm PST,
C.O.D.

Call for more information
(916) 924-8037
FAX (916) 972-9960

118

8051, 8096, 68HC11, 68332
SINGLE BOARD COMPUTERS



We feature a series of single board computers for process control applications. Each is available as a bare printed circuit board, or fully assembled and tested. Optional development software is also available. Please contact us to discuss your requirements and receive a literature package covering technical specs and pricing.

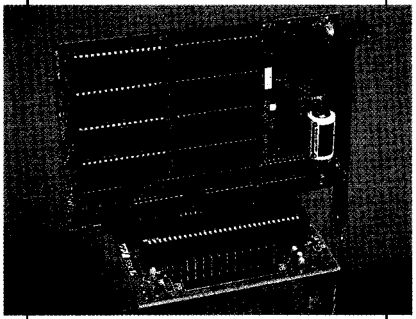
ALLEN SYSTEMS
2346 Brandon Road • Columbus, OH 43221
(614) 488-7122

VMAX®

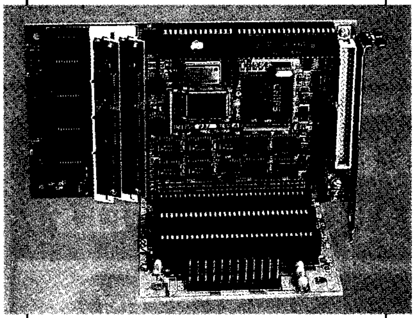
TOTAL PACKAGE PERFORMANCE

QUALITY PRODUCTS
RESPONSIVE SERVICE
RELIABLE DELIVERY

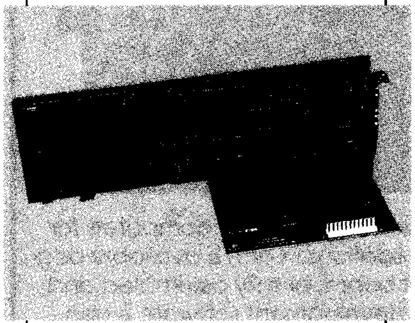
SOLID STATE DISK – $124*
½ Card 2 Disk Emulator
EPROM, Flash and/or SRAM
1 Meg Total, Bootable



TURBO XT WITH FLASH DISK – $266*
2 Serial, 1 Parallel Port
Up to 2 Flash Drives, 1 Meg Total
Software Included, 2 Meg DRAM



386DX Complete CPU – $1099*
40Mhz, IDE, FDC, SVGA
1Par, 2Ser (232/422/485), Cache
Solid State Disk-2 drives to 1.5 Meg



All Tempustech VMAX® products are PC Bus Compatible. Made in U.S.A., 30 Day Money Back Guarantee
*Qty. 10, 0 k, Call for Quantity Pricing

TEMPUSTECH, INC.
TEL: (800) 634-0701
FAX: (813) 643-4981

Fax for fast response!

295 Airport Road
Naples, FL 33942

#119

APPLICATIONS POTPOURRI

Here is a selection of application programs that illustrate the simplicity of the PLC solution to common control problems.

1. Push-on/push-off control

Two push-button switches, PB1(A0 input) and PB2 (Al input), control the load connected to YO output. Press any PB to switch the load on, and press any PB again to switch off. Any number of PBs may be added by ORing their transition inputs to the other inputs.

```
AXO.      Read PB1 input transition
AX1.      Read PB2 input transition
OR        Compute OR
YO.       Read output status
XOR       Compute exclusive OR
.YO       Write result to output
```

2. Pulsing output

A pulsing output is often desired over a steady output (e.g., flashing a warning beacon). When input A3 in ON, output Y1 pulses at a rate determined by PLC internal square wave signal CLKA.

```
A3.       Read control input
CLKA.     Read internal square wave
AND       Compute AND
.Y3       Write result to output
```

3. Real-time clock timers

Eight real-time timer channels with setable ON and OFF times are useful for controlling time-of-day events. Timer RTO in this example controls the load connected to output ZO.

```
TO.       Read timer RTO control bit
.zo       Write bit to output
```

4. Off delay timers

Off-delay timers are useful in "stretching" momentary signals. In this example, push-button PB1 on input B2 trigger delays timer DT1. The timer output, TQ1, controls output Z1. Timing starts when B2 input goes OFF. Output TQ1 remains ON until DT1 times out.

```
B2.       Read PB1 input
.TK1      Write to timer DT1 trigger input
TQ1.      Read timer DT1 output
.Z1       Write to output
```

would be to make use of the ubiquitous PC machine—desktop or portable-running a standard terminal emulator program and using the terminal as the programming interface. Most high-end PLCs and some low-end models now offer PC-based software to generate the application programs, but you may still have to purchase the software package.

The generation of the application program and its subsequent loading into the PLC are usually two separate tasks that may take several minutes to complete. You gain a large performance improvement if the program generation task is integrated into the PLC and can operate directly on the program storage buffer of the PLC. The need to transfer the program from the programmer to the PLC, and the time to do it, is eliminated because editing changes take effect immediately on the PLC.

If the program generation task [i.e., editing, interpreting, or compiling), execution code generation, and program loading is done directly in the PLC, then switching between the programming and operating modes of the PLC becomes easy. In fact, executing single instructions at a time and compiling the instructions incrementally in the program storage buffer becomes possible. The user may run the compiled program at full speed at any time. The benefits of this approach are evident during application program development, testing, and debugging, where you may easily enter and test short segments of a program with immediate feedback of the results.

Moving the program generation task out of the external programmer and into the PLC has the added benefit of cost savings due to programmer hardware elimination. Also, because the user interface is no longer constrained by hardware limitations, a far more powerful and user-friendly interface can be employed.

ARCHITECTURE

Figure 3 shows that describing precisely the requirements of a control logic scheme using a functional logic diagram is very easy, and I have said translating the diagram to the application program on any well-designed PLC should be equally easy. As a start, choose instruction words that describe exactly the function blocks and type of operations available. For example, you should be able to describe the AND, OR, NOT, and flip-flop functions as easily and intuitively as possible by the instruction mnemonics.

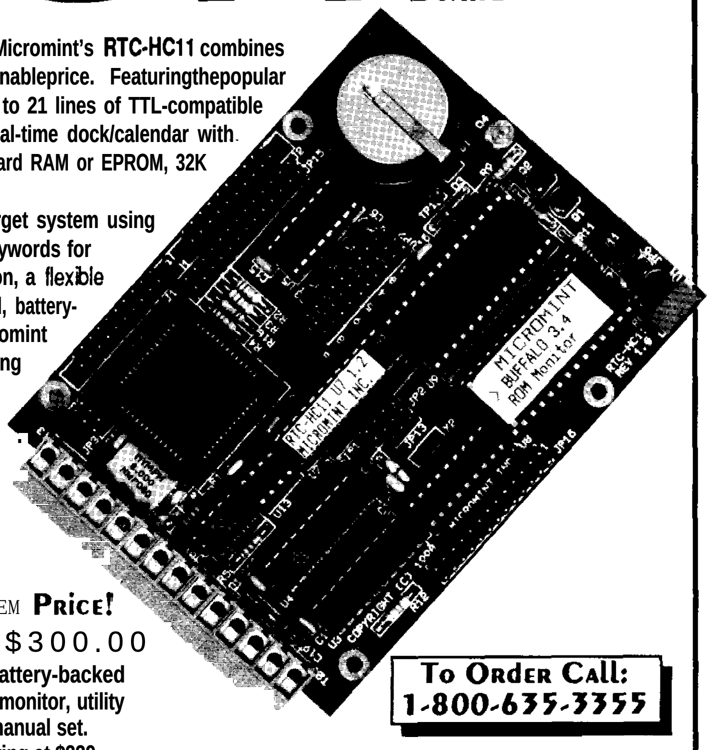
The PLC is a “virtual” machine, meaning it is a program that runs on the microcontroller board and emulates a logical machine. This emulated or virtual machine is designed to execute the PLC instruction “words,” which are routines written in the native language of the microcontroller. The architecture of the PLC machine should be designed for efficient bit-processing operations because logic control problems are characterized by these types of operations.

A very powerful technique for handling mathematical operations is to use a stack mechanism for data storage, manipulation, and transfer. A stack is a first-in, first-out data structure. New data is written onto the top of the stack, forcing existing data down. The data is read from the stack by sequentially popping it from the top of the stack, with the most recent data being retrieved first. If you employ a bit stack in your PLC, the operations for data manipulation are very straightforward and efficient. All logical operations are performed on the stack, and the results are left on the stack. Data can be transferred to and from the I/O table and the stack or the internal storage locations and the stack, or constants may be loaded to the stack. The model for the stack machine is very simple and the user can easily visualize and follow the stack operations.

RTC-HC11 PROCESSOR BOARD

Offering exceptional value in a single-board embedded controller, Micromint's RTC-HC11 combines allofthemost-asked-forfeatures intoasmall3.5"x4.5"packageat areasonableprice. Featuringthepopular Motorola MC68HC118-bit microcontroller, the RTC-HC1 1 gives you up to 21 lines of TTL-compatible ID; an 8-bit,8-channel analog-todigital converter; two serial ports; a real-time dock/calendar with battery backup; 512 bytes of nonvolatile EEPROM; and up to 64K of on-board RAM or EPROM, 32K of which can be battery backed.

Software development can be done directly on the RTC-HC11 target system using BASIC-1 1, an extremely fast integer BASIC interpreter with dedicated keywords for I/O port, A/D converter, timer, interrupt, and EEPROM support. In addition, a flexible configuration system allows a BASIC program to be saved in the on-board, battery-backed static RAM, and then automatically executed on power-up. Micromint alsooffersseveral hardwareandsoftwareoptionsfortheRTC-HC11 including the full line of RTC-series expansion boards as well as an assembler, ROM monitor, and C language cross-compiler.



Additional features include:
- Asynchronous serial port with fullduplex RS-232 and half-duplex RS-485 drivers
- 1-MHz synchronous serial port
- CPU watchdog security
- Low-power “sleep” mode
- 5-volt-only operation
- RTC stacking bus

SPECIAL DEVELOPMENT SYSTEM PRICE!
RTC-HC 11-DEV $300.00
Board w/ 8-bit ADC, EEPROM, 32K battery-backed RAM, Clock/Calendar, BASIC-1 1, ROM monitor, utility diskette with assembler and complete manual set.
Other configurations also available starting at $239.

To Order Call:
1-800-635-3355

MICROMINT, INC. 4 Park Street • Vernon, CT 06066 • (203) 871-6170 • Fax: (203) 872-2204

#120

The architectures of many of today's small PLCs are not readily apparent to the user, and the internal operation of the machine is normally hidden. If a bit stack architecture is used and a display of stack operations is provided, the user can follow the internal operations of the PLC and gain a better understanding of the machine. It is for these reasons that I specify a bit stack architecture for this PLC.

Careful thought should be given to the design of the instruction mnemonics with an aim to make the instructions intuitively easy for the user to remember and use. Most PLCs on the market will support at least a common set of logical operations, such as AND, OR, NOT, EXCLUSIVE-OR, and complex functions like timers and flip-flops. This PLC design should be no different, but because I use a stack architecture, I can adopt the following conventions to include the stack in the data transfer mnemonics.

Data transfer operations have two parts: the first part identifies the source and the second part identifies the destination of the data. The stack is represented by a dot, "." (shades of Forth) and labels are used to reference bit variables. Bit variables have a letter and number designation. Input bits use letters from the beginning of the alphabet, virtual bits use letters from the middle, and output bits use letters from the end. For example, input bit labels may be AO, B1, or C2; virtual bits may be UO, V1, or W2; and output bits may be X0, Y 1, or 22. The instruction mnemonic to read input bit A0 to the bit stack would be "AO." (pronounced "A0 dot") while the mnemonic to transfer the bit stack to output bit X2 would be ".X2" (pronounced "dot X2").

A logical operation on two variables removes the two topmost stack variables from the stack and places the result of the operation on the top of the stack. It has no source or destination parts because the stack is the implicit source and destination for the operands and result. Unary operations, such as NOT, SET, and CLEAR, simply replace the top of stack value with the new result.

Mnemonic	Action
.SO	Top of stack to flip-flop 0 set input
.RO	Top of stack to flip-flop 0 reset input
QO.	Flip-flop 0 output to top of stack

Figure 4—*The instructions for a flip-flop closely match the signals on the physical counterpart.*

Let me show how the functional diagram of Figure 3 would be translated to the application program, after first defining the instructions for the flip-flop in Figure 4.

The application program for the functional diagram in Figure 3 is shown in Listing la. Notice the program is nearly a direct intuitive translation of the functional diagram.

The result of the first AND gate is on the top of the bit stack at this point. When the following two inputs are read, the result is still stored on the stack, below the two new readings, as shown in Listing 1 b.

The top of stack now contains the result of (A6 AND AO). The next stack value is the previous result (A5 AND A3). The following two steps compute the logical OR of these values and send the result to the flip-flop SET input (see Listing Ic).

In a similar manner, you can write by inspection the instructions for the flip-flop reset input (see Listing 1d).

All the input conditions have now been read and the flip-flop output contains the control signal to send to the pump motor. The program shown in Listing 1 e simply transfers the results to the output bits.

Programming this type of PLC simply is a matter of setting down the control requirements on a functional diagram, then writing the application

Listing l--The *actual TILE* program for *the* functional diagram in Figure 3 may *look cryptic, but* is quite *straightforward.* Based *on* stack operations, *each* operator is *short but powerful.*

```
(a)
        A5.       Read HS-AUTO switch input
        A3.       Read LSL switch input
        AND       Compute (A5 AND A3)

(b)
        A6.       Read HS-MAN switch input
        AO.       Read START PB switch input
        AND       Compute (A6 and AO)

(c)
        OR        Read (A5 and A3) OR (A6 AND AO)
         .so      Write to flip-flop SET input

(d)
        A5.       Read HS-AUTO switch input
        A4.       Read LSH switch input
        NOT       Complement LSH value
        AND       Compute (A5 AND NOT.A4)
        AO.       Read STOP PB switch input
        NOT       Complement STOP PB value
        OR        Compute (A5 AND NOT.A4) OR NOT.AD
        A2.       Read OL1 switch input
        OR        Compute OR
         .RO      Write to flip-flop RESET input

(e)
        00.       Read flip-flop output
        .XO       Write to Motor output
        00.       Read flip-flop output
        NOT       Complement for Stop status
         .x2      Write to STOP indicator
        A2.       Read DL switch input
         .x3      Write to Overload indicator
```

program by inspection using mnemonics and labels that are intuitive. Once you use this method of programming, there is no going back to the clumsy ladder diagram method. The functional diagram serves to document the control scheme in an easily read and understood form, and it greatly eases maintenance and troubleshooting tasks.

In completing the design specifications of this improved PLC, you can add other functions to cater to special requirements and to improve the testing and debugging of application programs. By making the source code available, knowledgeable users may extend the instruction "words" of the PLC and customize it to special applications.

CONCLUSIONS

Machine control and automation projects benefit from the use of PLCs. As prices continue to fall, and performance and ease of use increases, more areas of application will open up. There is much room for improvement in small PLCs in the areas of architectural design, programming, and the user interface. I have defined the requirements for a small PLC that offers improvements in these areas.

A PLC designed to these specifications has been developed to run on an 803 1 microcontroller board. Named TILE, it supports up to 52 I/O points in its maximum configuration. TILE fits in a 16K EPROM that can be plugged into a controller board. For access to all the I/O functions, the board must support the required I/O interface devices, but TILE can operate with reduced I/O functions if the interface devices are missing, as in the cases where a simple evaluation is needed.

With TILE, anyone can easily tackle complex control and monitoring tasks in cost-sensitive application areas, such as home control and security monitoring. TILE is well suited for machine control and automation applications and for running test sequences on equipment that require frequent program changes. As a teaching tool, TILE is unbeatable with its ability to switch between program editing and full high-speed operating modes instantly. Its advanced stack architecture and carefully designed instruction words make programming an enjoyable task. ❑

Francis Lyn is a control systems engineer on petrochemical, power, and mining and refining types of industrial projects. He also does work with L.S. Electronic.

SOFTWARE

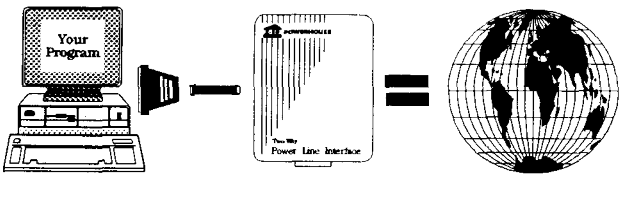
Software for this article is available from the Circuit Cellar BBS and on Software On Disk for this issue. Please see the end of "ConnecTime" in this issue for downloading and ordering information.

I R S

407 Very Useful
408 Moderately Useful
409 Not Useful

TW523
Power Line Interface
Developers Kit

Interface Your Computer To Transmit And Receive X-10 Codes Over Your AC Power Line.
Two-Way Communication.
Real Time Environment Control.



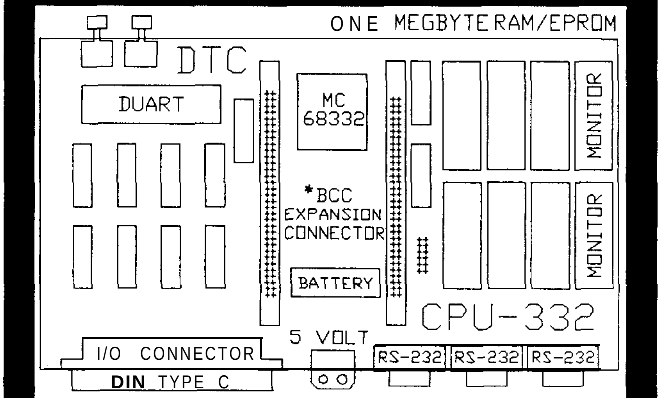
Kit Includes
TW523, Cable, Interface Connector (S/P)
Documentation. Source Code Supplied in "C", Pascal, BASIC or Run Time.
Disks 5.25in & 3.5in Format.

$75.00

Baran-Harper Group Inc.
Voice (416) 294-6473 BBS (416) 471-6776
Fax (416) 471-3730

#121

68332 68HC16
HIGH NOISE IMMUNITY



$595 QUANTITY ONE

Our FOUR layered CPU and I/O boards are designed for tough factory floor environments.
Custom design and board mfg. available.
Please call for specifics.

DTC

*Motorola "Business Card Computer"

TEL (510) 475-8147 • FAX (510) 489-5356
33476 Alvarado-Niles Blvd., Suite 6
Union City, CA 94587

#122

FEATURE ARTICLE

Dwayne Phillips

LZW Data Compression

In this age of information, greater emphasis is being placed on fitting more on a disk and shortening data transfer times. Find out the ins and outs of one of the more popular data compression methods.

One of the cliches I learned in college was "you never have enough disk space." I've found this little saying to be true all too often. One or two instances of having too much data and too little space, and I felt the ability to compress data was no longer a convenience and was instead a necessity. Several commercial products are available that meet this need, but how do they work? The README file in PKWARE's compression utility mentions Huffman coding and Ziv-Lempel-Welch schemes, but what do they do? To answer these questions and others, I will discuss some aspects of data compression and describe and demonstrate LZW data compression [1][2].

An issue that was arround long before computers, data compression allows you to store more data in less volume. Its first use was in communications because communicators needed to transmit information in as short a time as possible. One of the best known data compression codes is Samuel F. B. Morse's code, shown in Figure 1. The idea behind the Morse code is to represent frequently occurring characters by short codes and use longer codes for the other characters. For example, the letters e and *t* use only one dot or dash while the y uses four dots and dashes. The Morse code is one of history's best compression schemes because it worked. It worked so well that it almost killed Alexander Graham Bell's invention because telegraph operators could communicate faster than people could talk.

An excellent compression scheme introduced in the 1950s was the Huffman code [3]. Huffman created it with telecommunications (not computers) in mind. With Huffman coding, you use fewer bits to represent the most frequently occurring characters and more bits to represent other characters. The Huffman code provided minimum redundancy coding (i.e., the fewest average number of bits per character). In comparison, the ASCII code uses seven bits for each character. If a message contained 1000 *a's* and only one *b*, then there is no reason to use the same number of bits to represent the *a* and the *b*. Whereas, Huffman coding corrects this situation. It would represent the *a* with a single 0 and the ***b*** with a 10. The compressed message would only require 1000 x 1 + 1 x 2 = 1002 bits instead of 1001 x 8 = 8008 bits.

However, the Huffman code does have its problems. One is you must store the new code as a header to the compressed file. Because the frequency of characters is different in each file, you will have a different code for each file. Therefore, the decompression routine must read the code for the particular file before it can begin the decompression process. Another

A	.-	s	...
B	-...	T	-
C	-.-.	u	..-
D	-..	v	...-
E		W	.--
F	..-.	x	-..-
G	--.	Y	-_--
H	....	Z	- - .
	..	1	_----
J	.---	2	.---
K	-.-	3	...--
L	.-..	4	.-
M	__	5	.....
N	-.	6	-....
0	___	7	--. ..
P	.--.	8	---.
Q	--.-	9	----.
R	.-.	0	_____

Figure 1- Morse *Code compresses* data with *dots* and *dashes.*

Example message:

this is a **test<CR><LF>this** is a **test<CR><LF>**

The input characters and output codes are:

t	-116
h	-104
i	-105
s	- 115
_	- 32
is	- new code 256
_	-32
a	-97
_	- 32
t	-116
s	- 101
t	-116 -115
<CR>	- 10
<LF>	-13
th	- new code 256
is_	- new code 261
is_	- new code 261
a -	- new code 263
te	- new code 265
st	- new code 267
<CR><LF>	- new code 269

Figure **2—*In LZW compression, wde characters are assigned to variable-length strings lo achieve compression.***

problem is two passes must be made through a file to compress it. During the first pass, you count the occurrences of each character. Then, in the second pass, you code each character.

LZW COMPRESSION

LZW coding came along in the 1980s, and many commercial packages use it today. The name LZW comes from an article written by Terry Welch [2] and earlier work by J. Ziv and A. Lempel [1]. Note that UNISYS holds a patent on the LZW algorithm; you must talk to them before using it in a commercial package. Most people use some variation of the algorithm, and I do not know if the patent also covers these forms (see the last section, entitled Variations).

Among the well-known packages using LZW is the compress utility in UNIX systems. The compression packages from PKWARE also use LZW. The V.42bis modem protocol uses a form of LZW (the MNP-5 modem protocol uses a form of Huffman coding). The V.42bis is an example of things to come. You will see more and more instances of data compression in firmware and hardware in modems and hard disk controllers. V.42bis uses BTLZ [British Telecom Lempel-Ziv), another variation of LZW. CCITT chose BTLZ for their standard over MNP-5 and Hayes Adaptive Data Compression because it produced 3: 1 compression for text instead of 2:1 compression.

The difference between LZW and Huffman coding is the unit of information you code. With Huffman coding, you create a code for each character, but with LZW, you create a code for a string of characters. For example, if the input were *abcabcd, you* could output *abc#d. The #would* be a new code to represent *abc. You* create these codes as you read the input file. You don't need two passes to compress the file, nor do you need to read a header to decompress the file.

Let me show you a simple file containing *"this is a test<CR><LF> this is a test<CR><LF>."* Instead of sending characters, codes should be sent for the variable-length strings. If the string has not occurred previously, send the ASCII code for that character. When you reach a string that has occurred earlier, send a new code (in a moment, I'll explain how to create these codes). Figure 2 shows the input characters and the corresponding output codes.

By sending codes for the strings such as *th, is,* and *te, you* saved space. The input was 32 characters and the output was 22 codes. If you use 9 bits for each code, then the output **is 9 × 22 = 198 bits,** while the input was 256 bits. This compression was a good one for such a short file.

```
Initialize string table with single chars

String w = first input character

STEP:
    Read input character k
    If input file is empty Then
       output the code of string w
       EXIT program
    If wk is in string table Then
       w=wk
       go to STEP:
    Else wk is not in string table Then
       output the code of string w
       put wk in string table
       w = k
       go to STEP:
```

Figure 3—*The LZW compression algorithm is straightforward.*

Figure 3 shows the LZW compression algorithm. In the algorithm, w is a string and k is a character. The string table is a translation table that holds the strings and the codes (single numbers) to represent them. During compression, you read in a character k,

input	string w	char k	output	string table
t	t			
h	**t**	h	**t=116**	**256 = th = 116 h**
i	**h**	i	**h=104**	**257 = hi = 104 i**
s	i	s	**i=105**	**258 = is = 105 s**
_	s	_	**s=115**	**259 = s_ = 115 _**
i	_	**i**	_=32	**260 = _i = 32 i**
s	**i**	s	none	
	is	_	is=256	261 = is_ = 256 _
a		a	**=32**	**262 = _a = 32 a**
_	a	_	a=97	**263 = a_ = 97 _**
t	_	**t**	_=32	**264 = _t = 32 t**
e	**t**	e	**t=116**	**265 = te = 116 e**
s	**e**	s	**e=101**	**266 = es = 101 s**
t	s	**t**	**s=115**	**267 = st = 115 t**
CR	t	C R	**t=116**	**268 = tCR = 116 CR**
LF	CR	LF	**CR=10**	269 = CR LF = 10 LF
t	LF	**t**	**LF=13**	270 = **LFt** = 13 t
h	**t**	**h**	none	
i	**th**	i	**th=256**	271 = thi = 256 i
s	i	s	none	
_	is	_	none	
i	is_	**i**	is_=261	**272 = is_i = 261** i
s	i	s	none	
	is	_	none	
a	is_	a	is_=261	273 = **is_a** = 261 a
	a		none	
i	a-	t	a_=263	**274 = a_t = 263 t**
e	t	e	none	
s	te	s	**te=265**	275 = tes = 265 s
t	s	**t**	none	
CR	**st**	C R	**st=267**	276 = **stCR** = 267 CR
LF	**CR**	LF	none	
none	CRLF		**CRLF=269**	

Figure 4—*Taking the text in Figure 2, the compression algorithm assigns characters to certain stings to reduce the final size.*

```
DECOMPRESSION:
   code = oldcode = first input code
   k = code_of(code)
   output character k
   final-char  = k

NEXT CODE:
   code = next input code
   incode = code
   If input file is empty Then EXIT

   If code is not in the string table Then
      output character final-char
      code = oldcode
      incode = new_code_of(oldcode,
                final-char)
   end if

NEXT SYMBOL:
   If code_of(code) is a number +
                  character Then
      push k onto stack
      code = number pointed to by code
      go to NEXT SYMBOL:
   end if

   If code_of(code) is a single
               character = k Then
      output character k
      final-char = k

      do while stack is not empty
         output character on top of stack
         pop stack
      end do while stack is not empty

      put oldcode (number) and k
            (character) into string table
      oldcode = incode
      go to NEXT CODE:
   end if
```

Figure 5—*Decompression* is more *involved than compression,* but *takes less* time.

Figure **6**—*Looking al tie intermediate steps as the decompression* algorithm *works through a problem can aid in understanding* how *it works.*

input	oldcode	k	output	string table	
116		t	t		
104	116	h	h	256 = th	= 116 h
105	104	i	i	257 = hi	= 104 i
115	105	s	s	258 = is	= 105 s
32	115		_	259 = s_	= 115 _
256	32	i,s	is	260 = _i	= 32 i
32	256	_	_	261 =is_	= 258 _
97	32	a	a	262 = _a	= 32a
32	97	_	_	263 = a_	= 97 _
116	32	t	t	264 = _t	= 32 t
101	116	e	e	265 =te	= 116 e
115	101	s	s	266 = es	= 101 s
116	115	t	t	267 = st	= 115 t
10	116	CR	CR	268 = tCR	= 116 CR
13	10	LF	LF	269 = CRLF	= 10 LF
256	13	t,h	th	270 = LFt	= 13t
261	256	i,s,_	is_	271 = thi	= 256 i
261	261	i,s,_	is_	272 = is_i	= 261 i
263	261	a,_	a_	273 = is_a	= 261a
265	263	t,e	te	274 = a_t	= 263t
267	265	s,t	st	275 = tes	= 265s
269	267	CR,LF	CRLF	276 = stCR	= 267 CR

append it to the string w you read before, and see if this new string w k is in the string table. If it is, then you read in another character k and append it to the string. When you finally find a string w k that is not in the string table, enter it there and output (write to the compressed file) the code for the string w. The string table allows you to review all of the strings that have been read from the decompressed file and use them to build new, longer strings and codes to represent them.

Figure 4 shows how the compression algorithm created the codes for the text in Figure 2. You initialize the string table by placing the single characters in it. These occupy slots 0 to 255 of the string table. The algorithm inserts the strings created by reading the input file into the table starting at location 256. Figure 4 also shows that the algorithm outputs one code for each of the first five input characters and adds two character strings to the string table. When the algorithm encounters the word "is," it outputs the code 258 because this word is already in the string table. As the second ***"this is a test"*** is read, the algorithm finds several of these strings in the table. You added is_ to the string in location 261 during the first part of coding. Then, you output 261 twice during the last half of the coding. The output codes require less space than the input character strings, and compression occurs. If ***"this is a test"*** occurred a third time in the input, then a single code would represent four- and five-character strings.

Figure 5 shows the decompression algorithm. In Figure 5, code, oldcode, and incode are numbers, k and final_char are characters, and w is a string. In decompression, you read in a code from the compressed file. If the code represents a single character, you output that character to the decompressed file and go on. If the code represents a string, you recursively pull the last character off the string until the string itself becomes a character, and then output the characters. You build the string table as you go, as with the compression algorithm.

Listing l-Each *item in the* sting *table* has a *number and a* character.

```
/**************************************************
 *
 * This struct defines the items in the
 *  string table.
 *
 *
 *     code_num the number of the item in the table
 *
 *     code-char - the character appended onto the
 *        string in the table.
 *
**************************************************/

struct item {
   unsigned short code_num:
   char      code-char;
};
```

The decompression algorithm does not need the string table that the compression operation created. Decompression re-creates the table as it goes along. The first codes in any compressed file correspond to single characters. You already know the codes for the single characters (they're just the standard ASCII table), so you utilize these to build the codes for strings.

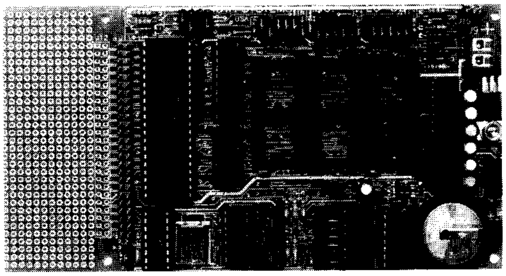
Figure 6 illustrates how the decompression algorithm reads in the codes produced in Figure 4, outputs the original characters, and re-creates the string table. Notice how the first codes read all represent single characters. Their character output is straight ASCII. While these simple codes come in, the 01 dcode and k build up the additions to the string table. When the code 258 [representing "is") comes in, it is in the string table. The N EXT SYMBOL section of the algorithm takes over and pushes the *s* down on the stack. After the NEXT SYMBOL section finishes, the algorithm outputs the *i*, then pops the *s* off the stack and

Listing 2—*Most of the functions needed for the "output the code of string w" segment of the compression algorithm* are shown here.

```
output_the_code(w,string_table,out_buffer,out_counter,out_file_desc)
   char     w[];
   int      *out_counter. out_file_desc;
   short    out_buffer[];
   struct   item string_table[];
{
   short n:
/* A. */
   find_string(w. string-table. &n);
/* B. */
   write_code(n. out-buffer. out-counter. out_file_desc);
}
/********************************************************/

find_string(w. string-table. n)
   char     w[];
   short    *n;
   struct   item string_table[];
{
   char w2[100]. x:
   int i. j, k. searching;
   w2[0]      = '\0';
   1          =  TABLE  1;
/* A. */
   x= last(w):
   searching = 1:
/* B. */
   while (searching) {
```

(continued)

EMBEDDED CONTROLLER
for Quick Development



The EC-32™ is a versatile 80C32 microcontroller board. It is ideal for quickly developing products, prototypes or test fixtures.

- 80C32 microcontroller (8051 compatible)
- BASIC-52 or MONITOR-52 available
- Program in C, BASIC or assembly language
- 8 to 92K RAM, EPROM or EEPROM
- Breadboard area and expansion bus
- RS-232 port and 12 digital I/O lines
- $100 for 11 MHz, $145 for 20 MHz

Iota Systems, Inc.
(702) 831-6302
POB 8987
Incline Village, NV 89450

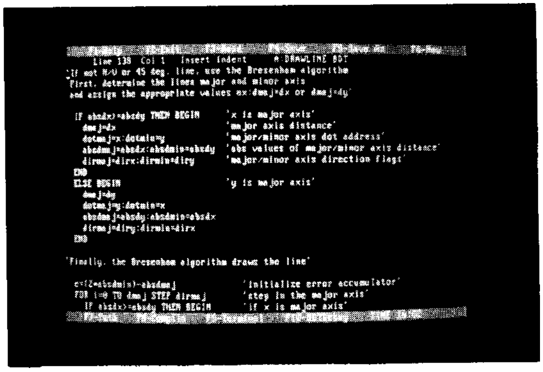
20 MHz Board Available

#124

BDT™ BASIC Developers Tool

ATTENTION: BASIC-52, BASIC- 180 and BASIC- 11 DEVELOPERS

Finally, **on** advanced development environment **for** BASIC **single-board computers. *BDT* combines all** the **tools you** need-Including ***Editor, Compiler, Debugger and Terminal*** emulator In **a** powerful, fast. easy-to-use. and **totally Integrated package.**



```
   Line 138  Col 1   Insert Indent     A:DRAWLINE.BDT
'If not H/V or 45 deg. line, use the Bresenham algorithm
'First, determine the lines major and minor axis
and assign the appropriate values ex:dmaj=dx or dmaj=dy'

 IF absdx>=absdy THEN BEGIN     'x is major axis'
   dmaj=dx                      'major axis distance'
   dotmaj=x:dotmin=y            'major/minor axis dot address'
   absdmaj=absdx:absdmin=absdy  'abs values of major/minor axis distance'
   dirmaj=dirx:dirmin=diry      'major/minor axis direction flags'
 END
 ELSE BEGIN                     'y is major axis'
   dmaj=dy
   dotmaj=y:dotmin=x
   absdmaj=absdy:absdmin=absdx
   dirmaj=diry:dirmin=dirx
 END

'Finally, the Bresenham algorithm draws the line'

 e=(2*absdmin)-absdmaj            'initialize error accumulator'
 FOR i=0 TO dmaj STEP dirmaj      'step in the major axis'
   IF absdx>=absdy THEN BEGIN     'if x is major axis'
```

Editor
. Configurable keystrokes and colors . Memory resident text (FAST!)
. Block move/copy/delete/read/write . Find & replace *•Auto-Indent*

Compiler
. Structured programs: DO/UNTIL. WHILE/WEND, BEGIN/END . No line numbers
. Up to 20-character variable and label names . Subroutine LOCAL variables
. Five types of comments (including multiline) stripped during *download*

Debugger
. Up lo 1000 *BREAK/PASSpoints . Execution PROFILE . Up to 40 WATCH variables*
. Integer variables WATCHable as DEC/HEX/BIN . All. or partial. array WATCH

Terminal
. Editor, file, and compile buffer *download to SBC . File capture from SBC*

Host PC ***Requirements***
. 5 12K . One disk drive . One serial port . Mono,C/E/V/GA . DOS 3.X-5.0

Individual versions ore **available** for BASIC-52 ***(BDT52)***, BASIC-180 ***(BDT180)*** and BASIC-I 1 ***(BDT11)*** SBCs @ $199. SPECIAL ***OFFER—"BDT-PAK"*** combines **all three versions @** $489 for a **savings** of over $100. Order today-offer **expires soon...**

MICRO FUTURE 40944 Cascado Place . Fremont. California 94539
(510) 657-0264 . FAX (510) 657-5441 . BBS (510) 657-5442

#12

Listing 2—*continued*

```
/* C. */
      if (x == string_table[i].code_char) I
         insert_-in_ front(w2, x);
         build_string(w2, string_table[i].code_num, string-table):
         if(strcmp(w2, w) == 0){
            searching = 0;
            *n = i;
         }/* ends if w2 == w */
         else {
            i--;
            w2[0] = '\0';


/* D. */
      else {
         i--;
         if (i < 0) {
            printf("\n\nLZW: ERROR did not find ",
                   "string in table\n");
            searching = 0:
            *n = 0:
         }
      }
   }
}
/*******************************************************/

insert_in_front(w, x)
   char w[], x;
```

(continued)

outputs it, too. The string table builds up and the process continues from there.

WRITING THE CODE

Listings 1 through 9 show some of the code that implements LZW data compression and decompression (the complete source and executable code are available on the Circuit Cellar BBS). The functions shown in the listings implement the key steps in the compression and decompression algorithms given in Figures 3 and 5. I wrote this program using Microsoft C 6.0, though nothing is unique to Microsoft, so the code should port quickly to other compilers and operating systems. The program is not intended to compete with commercial compression products, but is a demonstration of the ideas behind LZW compression. Compared to this code, commercial products execute much faster and with greater compression. On my 10-MHz '286 machine, compressing a 1000-character file takes around two minutes, and decompress-

GNU Cross Compilers

Are you looking for a 68xxx or 386/486 ANSI C MOSS compiler for DOS **OR UNIX?** Is complete source code to your compiler important to you? How about a complete Standard C library implementation, including floating point Emulation?

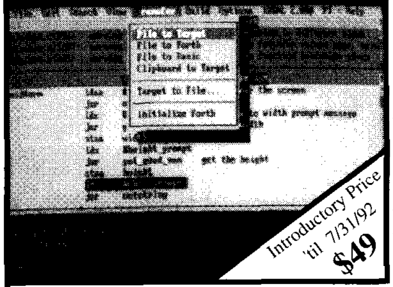
Hundred Acre Consulting is now offering GNU C support packages in several cross compiler configurations. Each package includes the GNU ANSI C compiler from the Free Software Foundation, a complete Standard C library with hooks for your own **file** system implementation, several support utilities, and one year of support at no additional charge. Platforms supported include MS/DOS. Sun 3 and Sun 4 with **SunOS** 4.1, and Interactive or **SCO** UNIX. Call today for more information!

GNU Support Package: $495

Hundred Acre Consulting
1280 Terminal Way, Suite 26, Reno NV89602
Phone: +1-702-329-9333
Email: info@pooh.com

#126

The **Quick** Way into Embedded Software Development



Integrated
Development
Environment
For (Almost)
Any
Language

IDEAL brings together all of the tools used to develop software for Single Board Computers (SBC) and blends them into a menu-driven windowed environment that speeds your development process.

- Automates the Compile-Assemble-Link-Download Cycle
- Works with any Compiler, Assembler, Linker, etc.
- Use the built-in Editor or link to your own Editor, it's easy
- Great for BASIC and FORTH interpreters too
- Interrupt driven serial communications to the SBC
- Swaps memory to make room for the largest tools
- Resizable, Zoomable Windows for Editing, Browsing and Target Communications
- Fully integrated mouse and keyboard control
- In use for 2 years at Major Universities and Corporations

Requires PC/XT/AT w/384K
Demo Available on BBS

30 day Money-Back Guarantee

creative applications engineering, inc.
P.O. Box 9524, Stanford, CA 93409

Phone (415) 494-2363
Compuserve: 74156,1207
BIX: ecarryer
BBS (415) 494-8363

#127

ing takes about one-fourth of that time. Disk I/O is not a factor in performance.

Listing 1 shows the data structure used to implement the string table. Each item in the string table will have a number and a character associated with it. This feature takes care of the number and character pairs shown in the far right of Figure 4.

Listing 2 shows most of the functions needed for the ***output the code of string*** w segment of the compression algorithm (Figure 3). The function output_the_code has two steps. First, it finds the code that represents the string w. Then, it writes the code. The find_string function does the slow process of searching through the string table to find the string and the number representing it. find_string sets the last character of w to x (A) and searches for x using the characters in the table (B). Then it builds up the string and compares it to w(C). find_stringcallsthenexttwo functions of Listing 2, *insert_in_front* and build_string. insert_

Listing 2—*continued*

```
{
   char temp:
   int i. j. not-finished;
/* A. */
   i = 0;
   not-finished = 1;
   while (not-finished) {
      if (w[i] == '\0')
         not-finished = 0:
      else
         i++;
.
/* B. */
   w[i+1] = '\0';
   for (j=i;j>0;  --)
      w[j] = w[j-1]
/* c. */
   w[0] = x:
}
/*********************************************************/

build_string(w, number. string-table)
   char   w[];
   short  number;
   struct item string_table[];
{
/* A. */
   if(string_table[number].code_char != '\0'){
      insert_in_front(w, string_table[number].code_char);
```

(continued)

The *only* 8051/52 BASIC compiler that is

100 % BASIC 52 Compatible

and

has full floating point, integer, byte & bit variables.

- Memory *mapped* variables
- *In-line assembly language* option
- *Compile time switch to select 805 1/803 1 or 8052/8032 CPUs*
- *Compatible with any RAM or ROM memory mapping*
- *Runs up to 50 times faster than the MCS BASIC-52 interpreter.*
- *Includes Binary Technology's SXA51 cross-assembler & hex file manip. util.*
- *Extensive documentation*
- *Tutorial included*
- *Runs on IBM-PC/XT or compa tibile*
- *Compatible with all 8051 variants*
- ***BXC51 $ 295.***

603-469-3232
FAX: 603-469-3530

Binary Technology, Inc.
Main Street • P.O. Box 67 • Meriden, NH 03770

#12

▶ Hard-to-find stuff ◀

CS212 SART I/O network chip	8.10
MC1 45030 IR encoder/decoder	9.70
MT8809 8x8 analog crosspoint	10.30
TW523 XI 0 2-way interface	45.00
PL513 XI 0 bans-only interface	8.00
IL300 *linear* optoisolator	8.80
Sharp IS1 U60 38 kHz IR rec	4.20
Sharp GP1U52X 40 kHz IR rec	6.75
LD273 dual IR LED *(bright)*	2.10
16C55 logic analyzer w/DRAM	19.00

Experimenters: Our 8031-LINK 'Chip Sack" has the ICs you need to build an HCS II COMM-LINK for $34.00...

UPS Ground/2nd day/next day $6/8/16 to 46 US states, COD add $4. Check or MO only, no credit cards or POs. CT residents add 6% sales tax. Quantity discounts!

Pure Unobtainium

▶ Your unusual parts source ◀
89 Burbank Road
Tolland, CT 06084-2416
FAX/voice (203) 870-9304

#129

Liiting 2—*continued*

```
/* B. */
     if(string_table[number].code_num != 0)
        build_string(w,
                     string_table[number] .code_num,
                     string-table):
  }
}
/*******************************************************/

write_code(n, out-buffer. out_counter. out-file_desc)
   int     *out_counter. out_file_desc:
   short   n. out_buffer[]:
{
   int bytes_wri tten,i:
/* A. */
   if(*out_counter >= LENGTH2){
      bytes-written = short_output(out_buffer,
                                   out_file_desc,
                                   out_counter);
      LZWTEST(printf("\nLZW: TEST: wrote %d bytes',
                     bytes_wri tten);)
      *out_counter = 0:
      for(i=0; i<LENGTH2: i++)
         out_buffer[i] = 0:
   }
/* B. */
   out_bufferC*out_counterl = n:
   *out_counter = *out_counter + 1;
   LZWTEST(printf("\nLZW: TEST: output %3d - out coun .er = %3d",
           n, *out_counter); )
```

in-front puts a character into the beginning of a string.

The build_string function uses the recursive nature of the string table to build up a string. You give it a place in the string table pointed to by number. If there is a character in the string table, then put that character into the beginning of the string w (A). If that place in the string table points to another place, then build_string calls itself recursively (B). build_string keeps calling itself until the place in the string table no longer points to any strings.

The final function of Listing 2 is write_code, which puts the code into an array. If the array is full [A), it calls the short_output function for writing to disk.

Listing 3 shows the is_present function, which implements the ***If wk is in string table*** segment of the compression algorithm (Figure 3). This function is similar to find_string shown earlier. It searches the string table looking for the character k. If it finds k (B), it builds up the string w2

DSP

HARDWARE

Check out our complete line of DSP boards based on powerful floating-point processors like the AT&T DSP32c (25 MFlops) and the Analog Devices ADSP-21020 (75 MFlops). Several analog interface modules are available. DSP boards start at just $995.

SOFTWARE

We have everything you need to do DSP software development, including C compilers, assemblers, source-level debuggers, algorithm development tools, and many example programs. Data can be transferred between the DSP board and host at up to 3 Mbytes/sec with the host interface library (source code included).

SOLUTIONS

Call our friendly, knowledgeable staff to discuss your applications and we'll show you how easy it is to take advantage of DSP technology.

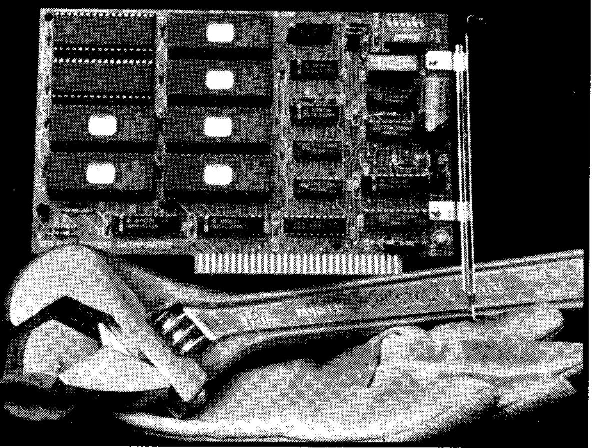
800-848-0436

BITWARE
RESEARCH SYSTEMS
400 East Pratt Street • 8th Floor • Baltimore • MD • 21202
More Info: 800-848-0436 • FAX: 410-783-7375

#130

SOLID STATE DISK

- BOOTABLE
- VERSATILE
- SIMPLE
- COMPATIBLE
- DURABLE
- FAST
- RELIABLE
- ANOTHER SEALEVEL/ANNABOOKS INNOVATION



SEALEVEL
COMMUNICATIONS & I/O

SEALEVEL SYSTEMS INC.
PO BOX 830
LIBERTY, SC 29857
(803) 843-4343

#131

and compares that with w. is_present and find_string are slow because they call build-string each time they find a match in the string table. There are faster ways for finding a given string in the table, and I describe these modifications in the next section.

Listing 4 shows the insert_into_table function, which implements the *put wk in string table* segment of the compression algorithm. This function finds the first open place in the string table (A). It then finds the number that represents the string w (B) and places this number and the character k into the open place in the table (C). Listings 1 through 4 contain the key functions for the compression process.

Listings 5 through 9 show the key functions for the decompression process. Listing 5 shows the write_character-k function, which performs the output task given in several places in the decompression algorithm of Figure 5. This function puts the output character into a character array for writing to disk. If the array is full, then it is written to disk (A).

Listing 6 shows the function next_code, which implements the *incode = new_code_of(oldcode, final_char)* portion of the decompression algorithm. This function looks for the place in the string table where you would put the next code by finding the first open place in the string table.

Listing 7 shows the function next_symbol, which implements the NEXT ***SYMBOL*** portion of the decompression algorithm. This function takes a string pointed to by code, pulls characters off the string, and places them on a stack for later output. It first finds the place in the string table pointed to by code (A). If the character in that place is **nonnull,** it pushes that character onto the stack and calls itself using the new place as the code (B). This process continues until the code represents a single character.

Listing 8 shows the code-of function, which implements the ***code-of (code)*** statement shown in several places in the decompression

Listing **3—*The is_present*** *functions searches the sting fable* **for** *the given sting.*

```
is_present(w, k. string-table)
   char   w[]. k;
   struct item string_table[];
{
   int i. result. searching:
   char w2[100];

   w2[0]     = '\0';
   result  =  0:
   i          = TABLE  1:
   searching = 1;
/* A. */
   while (searching) {
/* B. */
      if (k == string_table[i].code_char) {
         build_string(w2, string_table[i].code_num, string-table):
         if (strcmp(w, w2) == 0){
            result    = 1;
            searching = 0;
         }/* ends if w2 == w */
         else {
            i--;
            w2[0] = '\0';
         }/* ends else w != w2 */
      } /* ends if found a match for k */
/* c. */
      else {
         i--;
         if (i < 0) searching = 0;
      }
   }
   return (result):
}
```

Listing *4-The* ***insert_into_table function implements*** *the 'put* ***wk*** *in* ***string table" portion*** of We ***compression algorithm.***

```
insert_into_table(w, k. string-table)
   char   w[]. k:
   struct item string_table[];
{
   int i, j.  searching:
   short  s;

   i          = TABLE -1;
   searching = 1:
/* A. */
   while (searching) {
      if (string_table[i].code_char != '\0')
         searching = 0:
      else {
         i--;
         if (i < 0)
            printf("\nLZW: ERROR- Table full--cannot insert");
      }
   }
/* B. */
   find_string(w, string-table. &s);
/* c. */
   string_table[i+1].code_num = s:
   string_table[i+1l.code_char = k;
   LZWTEST(printf("\nLZW: TEST: inserted string %d",s);)
   LZWTEST(printf(" and char %c into table", k);)
}
```

algorithm. This function goes to the place in the string table pointed to by code and returns the code and character in that place.

Listing 9 shows the put_in_table function, which implements the *put oldcode (number) and k (character) into string table* portion of the decompression algorithm. This function searches for the first open place in the string table (A] and puts the number and character into that place (B).

As I stated earlier, this code executes slowly. The decompression process is much quicker than compression because it does not search the table by comparing strings. The compression process is especially slow because of the way I implement and search the string table. The functions find-string and is-present both search through the string table by building up every possible string that might match the target of the search. Making these searches more efficient would save time. I leave this exercise for you.

Listing 5—*write_character_k* **performs the output** *task given in several* **places** *in* **the** *decompression* **algorithm. It** *pub* **the output** *character into an array* **for** *writing* **to** *disk.*

```
write_character_k(k, out-buffer, out_counter, out_file_desc)

   char k, out_buffer[];
   int  *out_counter, out_file_desc;

{
   int bytes-written:

/* A. */

   if (*out_counter >= LENGTH1) {

      bytes-written = write(out_file_desc,
                            out-buffer,
                            LENGTH1);

      LZWTEST(printf("\nLZW: TEST: wrote %d bytes".
                 bytes-written): )
      *out-counter = 0:


/* B. */

   out_buffer[*out_counter] = k;
   *out-counter  = *out_counter + 1;
}
```

The $595 Solution
to 8051 System Development

PDK51

- *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.*

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.

PDK51 PLUS includes everything in the PDK51 plus Vers. 3 of our popular BXC51 8051/8052 BASIC compiler—**$800.**

Call Now! 603-469-3232 or FAX 603-469-3530

Binary Technology, Inc.
Main Street . PO. Box 67 . Meriden, NH 03770

#133

DC/CAD

introducing.. .

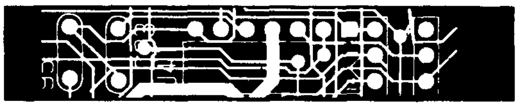
THE TERMINATOR

Super High Density Router
(Complete with Schematic & PCB EDITOR)
Features the following powerful **algorithm & capability:**

- **Rip - up and Retry**
- Pre-routing of SMT components
- Real-Time via **minimization**
- Real-Tie clean up passes
- **User** defined **strategies**
- **Window** 3.0 **capability** as **DOS** Task
- 1-mil Autoplacer and **Autopanning**
- Two-way Gerber and DXF
- **Automatic** Ground Plane **w/** Cross-Hatching
- Complete **w/** Schematic **&** Dcly Libraries
- Optional simulation capability & protected mode for 386 users

Call for
FREE DEMO DISK
price range:
$295 - $1495

* PCB LAYOUT SERVICE AT LOW COST *
LEASE PROGRAM & SITE LICENSE AVAILABLE



Design
Computation

1771 State Highway 34
Farmingdale, NJ 07727
(908) 681 - 7700 • (908) 681 - 8733 (FAX)

" DC/CAD ... The focal point of future CAD market "

#134

Listing 6—*next_code finds the place in the string table where he next code would go by finding the first open place in he string table.*

```
next_code(oldcode, final-char. string-table)

   char final-char:
   short oldcode;
   struct item string_table[];
{
   int i, new-code. searching:

/* A. */

   searching = 1:
   i = TABLE 1;
   while (searching) {
      if (string_table[i].code_char != '\0')
         searching = 0:
      else
         i--;
   }

/* B. */

   new-code = i+1;
   LZWTEST(printf("\nLZW: TEST: inserted into table'):)
   LZWTEST(printf(" new-code = %d", new-code): )
   return(new_code);
}
```

VARIATIONS

The LZW algorithm has several variations. As with any algorithm, you can tinker with portions of it and improve on the speed. Most work on LZW has to do with the implementation of the string table and how you search it [4]. You can investigate any searchable data structure and any method of implementing it. The possibilities are endless.

You can also investigate the amount of compression. The program I describe here represents the codes with 12 bits, which gives you 4096 possible codes. What if you need more or fewer codes? The program could automatically examine the input file, "guess" at the number of codes needed, and use this number. Another idea is to use 9 bits for the first 5 12 codes, switch to 10 bits for codes 512 through 1023, switch to 11 bits for codes 1024 through 2047, and so forth. How would you keep track of the bits? How would you pack and unpack them? How could you do all of these things quickly and efficiently?

1802 6800 8041 COP400
3870 6801 8048 COP800
4004 6805 8051 SUPER8
6301 6809 8085 Z8
6502 68HC11 8096 Z80
64180 78C10 89700 MORE...

Cross-l 6 Meta-Assembler: US$99 / CN$119

XDASM Cross-Disassembler: US$249/CN$299

- Both MS-DOS products include support for ALL of the above processor families.
- EPROM emulators and Forth compilers too!
- Request our catalog.
- Credit cards are billed in Canadian dollars (CN$).
- Canadian residents please add 7% G.S.T.

Universal Cross-Assemblers

P.O. Box 6158
Saint John, NB, Canada
E2L 4R6
Voice / Fax: (506) 847-0681

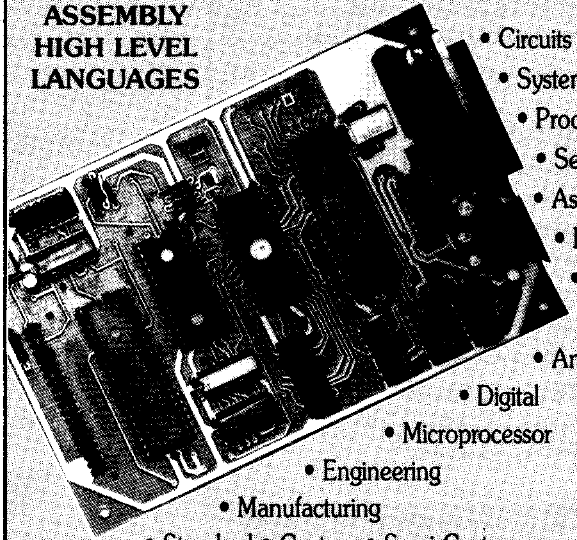
MasterCard AMERICAN EXPRESS VISA

135

MAGNUM OPUS
CUSTOM ELECTRONIC ENGINEERING

SINGLE BOARD COMPUTERS
EPROM PROGRAMMERS

ASSEMBLY
HIGH LEVEL
LANGUAGES



• Circuits
• Systems
• Products
• Services
• Assembly
• Repair
• Test

• Analog
• Digital
• Microprocessor
• Engineering
• Manufacturing
• Standard • Custom • Semi-Custom

284 Kennedy St • Iselin, NJ 08830
(908) 283-4925

#136

I invite you to experiment with the code. There is room for improvement in data compression-both in implementation details and completely new techniques. Remember, no one ever has enough disk space. ❑

Dwayne Phillips is a computer and electronics engineer with the U.S. Department of Defense. He has a Ph.D. in Electrical and Computer *Engineering from Louisiana State University. His interests include computer vision, artificial intelligence, software engineering, and programming languages.*

SOFTWARE

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

REFERENCES

1. J. Ziv and A. Lempel, "Compression of Individual Sequences via Variable-Rate Coding," *IEEE Transactions on Information Theory,* vol. 24, no. 5, (September 1978).

2. Terry Welch, "A Technique for High-Performance Data Compression," *IEEE Computer,* vol. 17, no. 6, (June 1984).

3. David Huffman, "A Method for the Construction of Minimum Redundancy Codes," *Proceedings of the IRE,* vol. 40, no. 9, (1952): 1098-l 101.

4. Timothy C. Bell, "Better OPM/L Text Compression," *IEEE Transactions on Communications,* vol. 34, no. 12, (December 1986): 1176-l 182.

I R S

410 Very Useful
411 Moderately Useful
412 Not Useful

Listing *7—next_symbol takes a string, pulls characters off, and puts them on a stack*

```
next_symbol(code, string-table. stack. stack_pointer)
   char   stack[], *stack-pointer:
   short  *code:
   struct item string_table[];
{
   short code2, s:
   char  c;
/* A. */
   code2 = *code:
   code_of(code2, &s, &c, string-table)
/* B. */
   if (s != 0){
      push(c, stack. stack_pointer);
      if (s > 255) {
           *code = s;
           next_symbol(code, string-table. stack. stack_pointer);
      }
/* C. */
      else
         *code = s:
   }/* ends if s != 0 */
}
```

Listing *8—code_of goes to a specified* **place** *in the string passed to it and returns the code and character in that place.*

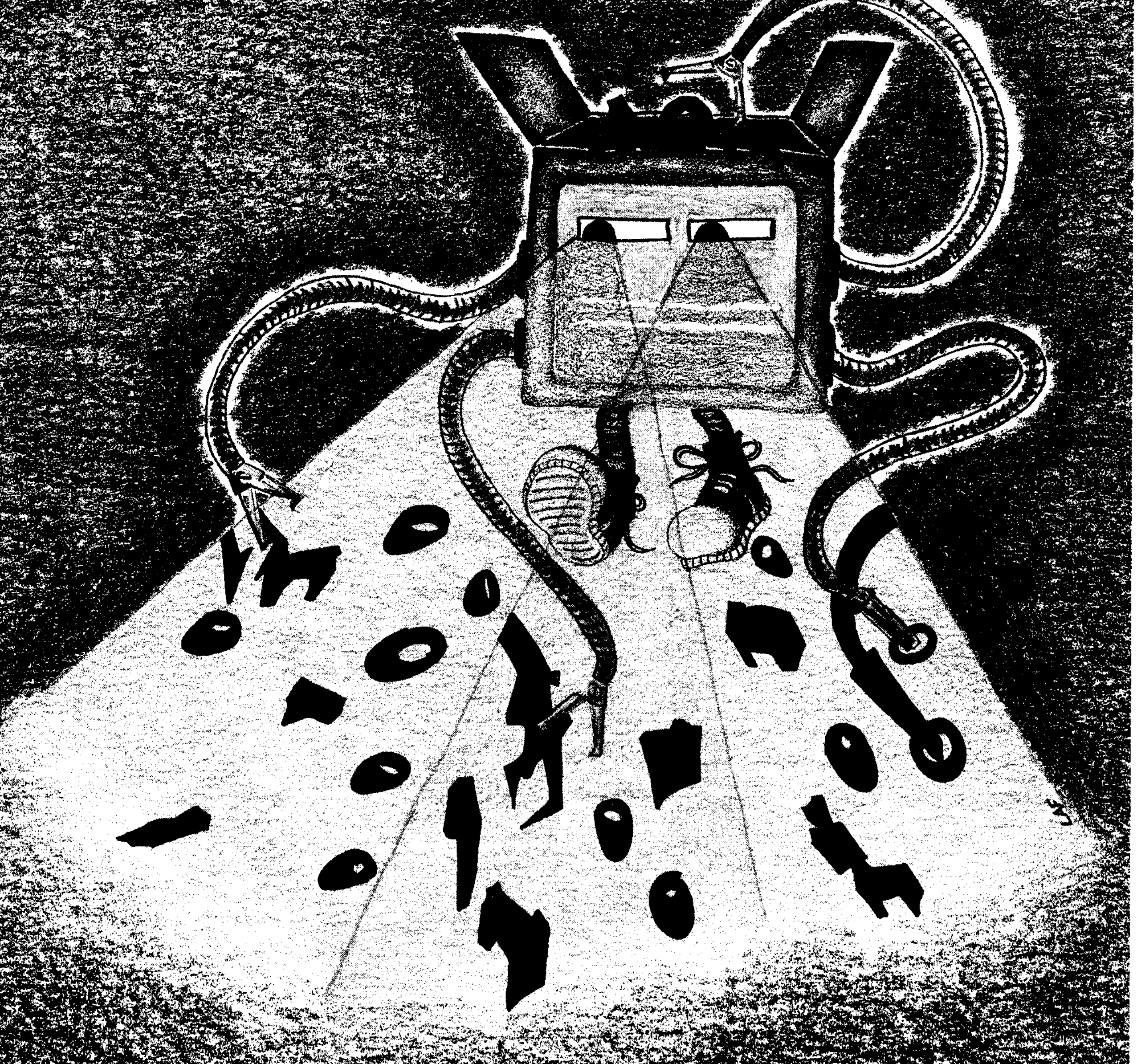
```
code_of(code, s. x. string-table)
   char   *x;
   short  code, *s;
   struct item string_table[];
{
   *x = string_table[code].code_char;
   *s = string_table[code].code_num;
}
```

Listing 9—*put_in_table searches for the first open place in the string table and puts the given number and character into that place.*

```
put_in_table(n, c. string-table)
   char   c:
   short  n:
   struct item string_table[];
{
   int i, searching;
/* A. */
   searching = 1;
   i = TABLE 1;
   while (searching) {
      if (string_table[i] .code_char != '\0')
         searching = 0;
      else
         i--;
   }
/* B. */
   string_table[i+1] code_num = n;
   string_tableCi+ll  code-char = c;
}
```

SPECIAL SECTION
Embedded Sensors & Storage

50 The Elements of a Data Logger
by John Dybowski



The Elements of a Data Logger

SPECIAL SECTION

John Dybowski

Batteries, real-time clocks, power-control circuits, A/D converters—they are all just as important as the microcontroller in a data logger. Find out what to look for when selecting data logger components.

Collecting analog data can be accomplished in a variety of ways. You may use a dedicated PC equipped with an ADC card, or elect to roll your own and lash an integrated circuit ADC to a general-purpose microcontroller card. At the very least, the data collection system should be tolerant of power disturbances. Better yet, it should have the capability to run exclusively from a battery because AC power may not be readily available where you want to perform the analog measurements. Having determined that battery operation is a good thing, you can dismiss the PC-based approach immediately, but how long can you run a microcontroller-based logger off a battery? Four hours? Eight hours? Hardly an attractive proposition.

The trick to battery operation, as applied to low-power data logging, is to run the power-hungry circuit elements only during the acquisition and data processing phase. Burning up limited battery power while waiting for something to happen makes no sense.

In such a configuration, continuously running low-power timing circuitry is used to issue a wake-up call to the logic and converter elements at predetermined intervals. On emerging from reset, the controller reads the ADC; processes, time stamps, and stores the data; and rearms the timing circuitry for the next reading before shutting down the main power. You end up with a standard microcontroller core with nonvolatile RAM, an ADC, an W-232 interface, a timekeeper, a power manager, and a battery power source.

I have designed and constructed two different data loggers that incorporate these basic elements. Differing in the controller and power supply sections, these loggers use the same ADC, timekeeper, and communications interface, and being based on 8031-style controllers, both operate under control of what is basically the same program. I'd like to describe some of the more common points before looking at the actual device implementation.

THE BATTERY

The main power for the data loggers I designed is sourced from a 9-volt battery, either a NiCd rechargeable or a standard alkaline. The 9-volt NiCd battery is typically made up of either six or seven 1.2-volt cells providing an overall output of 7.2 or 8.4 volts, respectively. I selected the 8.4-volt battery for my logger. The primary difference between NiCds and alkalines (aside from the fact that the NiCd is rechargeable, of course) is that the NiCd is capable of supplying 100 mAH where an alkaline battery has five times that capacity, or 500 mAH. Before moving along, let me quickly describe the care and feeding of NiCds.

The charge rate of a NiCd battery is usually expressed in multiples of a C rate. The C rate is defined as the rate in amperes or milliamperes numerically equal to the capacity rating of the cell. For example, for a cell with a 100-mAH capacity, the rate of 1C is 100 mA. These C rate charging currents can be characterized into such descriptive terms as Standard Charge, Quick Charge, and Fast Charge.

Overcharge is the normal continued application of charging current to a battery after it has reached its maximum state of charge. Charging cells that are already fully charged causes a rise in oxygen pressure within the cells. Along with the rise in pressure comes a corresponding rise in cell temperature. Standard cells may be overcharged at rates up to 0.1C. Quick-Charge cells and Fast-Charge cells, which are designed to withstand higher overcharge rates, are normally charged at 0.2C and 1C, respectively. At high charge rates, a control mecha-

Listing 1—*The MC146818A real-time clock can be used as an interval timer.*

```
; R3 CONTAINS THE SECONDS LOGGING INTERVAL INPUT BY CALLER
; R7 CONTAINS THE CURRENT SECONDS RETURNED BY GET-TIME

         CALL      GET-TIME           ; Get the real time
         MOV       A,R3               ; Seconds are in R3
         ADD       A,R7               ; Add the logging interval
         CJLE      A,#59,L?STB1       ; Jump if no adjustment required
         CLR       C
         SUBB      A.#60              ; Adjust
L?STB1:  MOV       R5,#OFFH           ; Set hours to don't care
         MOV       R6,#OFFH           ; Set minutes to don't care
         MOV       R7.A               ; Set the seconds value
         CALL      PUT-ALARM          ; Set the alarm registers
```

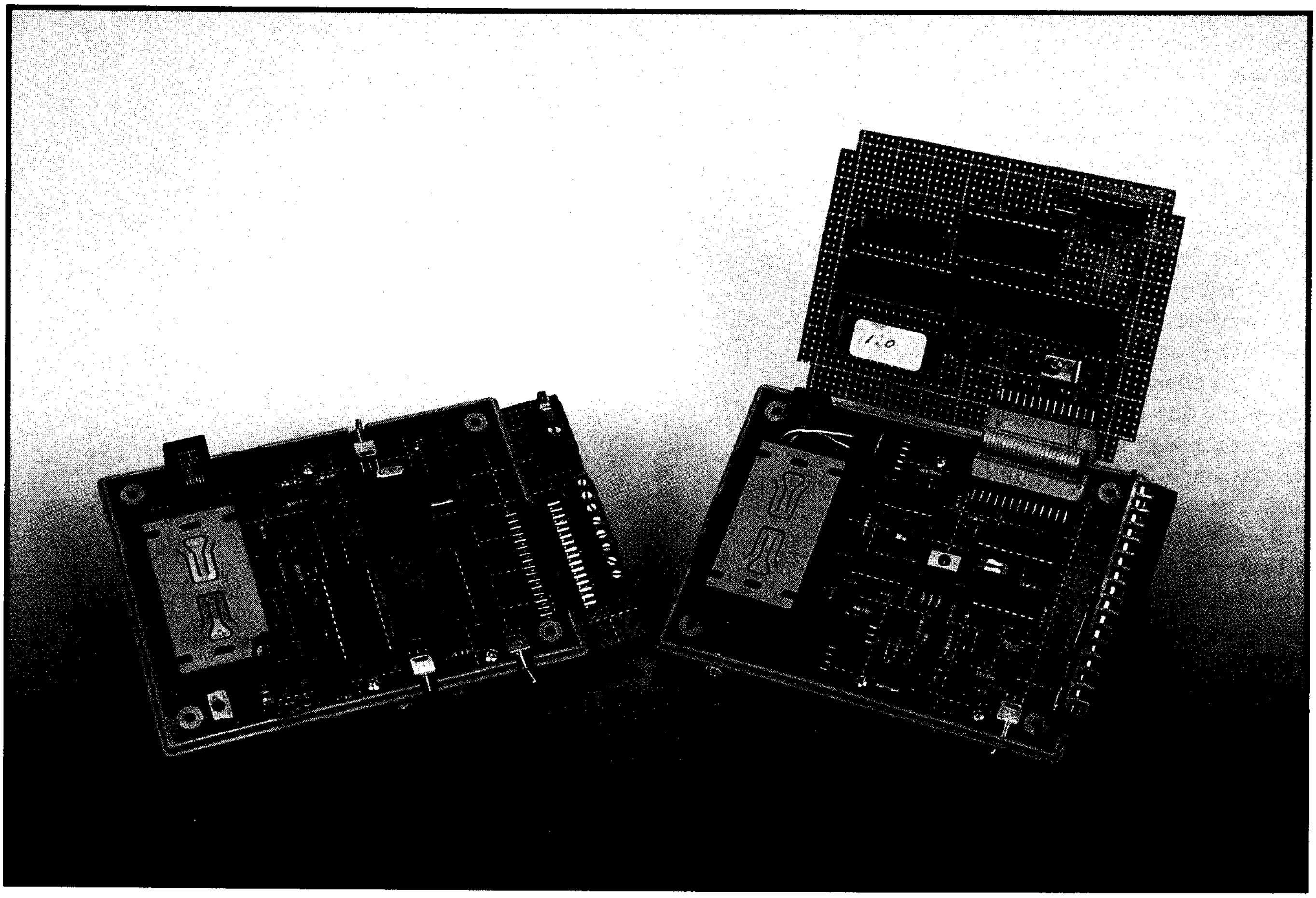
nism is required to reduce the charge current to 0.1C for standard batteries once a full charge has been attained. Various methods may be used to accomplish this reduction, such as timed control, pressure sensing control, or temperature sensing control. Charging at the 0.1 C rate does not require these controls, and in the interest of simplicity my data loggers utilize 0.1 C rate charging.

SYSTEM TIMING

The system timer's function is to peroidically issue a signal to power up the data logger. This timer runs continuously so must consume as little current as possible. Because a real-time clock is required for the time stamping of collected data, why not also use the RTC as the interval timer?

There are many timepieces available for use with microcontrollers, many of which feature time-base outputs. Unfortunately, most of the time bases are fixed or allow only the selection of a limited number of output frequencies. However, a part does exist that provides more than enough flexibility for our needs. The MC146818A, available from Motorola, Hitachi, and others, combines a complete time-of-day clock with an alarm, a 100-year calendar, and 50 bytes of static RAM.

The MC146818A has three time-base outputs: CKOUT, SQW, and ● IRQ. CKOUT is the time-base frequency divided by 1 or 4, selected by the CKFS pin. The SQW pin provides more flexibility and can output a signal from one of 15 taps provided by the 22 internal divider stages, and it is software programmable. ‘IRQ is an open-drain output that goes low in response to an internal interrupt condition. The RTC includes three separate fully automatic sources of interrupts. These sources include the “update ended” interrupt,



Two approaches to the same data logger: one based on an 80C31 (right) and another based on a DS5000 (left). The DS5000-based system is obvious/y smaller and has the advantage of more digital I/O.

the periodic interrupt, and the alarm interrupt. An alarm interrupt sounds like just the thing to me. Now let me describe how it works.

Alarm bytes exist for Seconds Alarm, Minutes Alarm, and Hours Alarm. The three alarm bytes may be used in two ways. First, when the program sets the alarm time, the alarm interrupt is initiated at the specified time each day if the alarm enable bit is high. The second usage is to insert a "don't care" code (hex CO through FF) into one or more of the bytes. An alarm interrupt each hour is created with a "don't care" code in the Hours Alarm location. Similarly, an alarm is generated every minute with "don't cares" in the minutes alarm location, and so forth. With a little software, the RTC's alarming capability can be used to generate signals at intervals from once a second to once a day.

The procedure used to perform interval timing using the MC146818A consists of the following steps. At configuration time, store the logging interval value along with the increment (seconds, minutes, or hours) in nonvolatile RAM. The final action the controller performs following the acquisition of data is to set the alarm for the next interrupt. Begin by reading the current time, then add the logging interval to the appropriate element, set the remaining elements to "don't cares," and write the new values to the alarm registers. If after the addition, the value exceeds the limit for that particular element, the limit is subtracted. Say I am setting the seconds registers and the current seconds are 55 and the logging interval is 10. The calculation results in 65 (55 + 10 = 65). The limit in this case is 60, so I subtract 60, and the new value is 5. Setting the minutes and hours to "don't cares" results in the interrupt occurring when the seconds equal 5; the desired result.

The MC 1468 18A allows either BCD or binary representation of data. Using the binary mode simplifies the math, and the binary-to-ASCII conversion routines are required anyway to handle the other binary entities, such as the ADC data, so this aspect is not a problem.

Listing 2-The *ADC08138 ADC is used for analog inputs. NOPs are used to maintain a 40% to 60% duty cycle on the dock.*

```
; INPUT: ACC CONTAINS ADC CHANNEL
; OUTPUT: ACC CONTAINS ADC DATA
.
READ_ADC    PRDC
            CALL     XLATE           ; SET UP BIT PATTERN
            CLR      ADC_CLOCK       ; DROP CLOCK
            CLR      ADC_CS          ; ASSERT SELECT
            MOV      R1,#5           ; 5 BITS TO SEND

; WRITE ADC ADDRESS TO START CONVERSION
:
L?RA1:      RLC      A               ; POSITION DATA BIT
            CLR      ADC_CLOCK
            NDP
            MOV      ADC_DATA.C      : MOVE TO PORT PIN
            SETB     ADC_CLOCK       ; CLOCK ON RISING EDGE
            DJNZ     R1,L?RA1        ; LOOP UNTIL DONE
            SETB     ADC_DATA        ; PREPARE FOR INPUT
            MDV      R1 ,#9          ; 8 BITS+SAMPLE TIME

 : ADDRESS IS SET, NOW READ CONVERSION RESULT
.
L?RA2:      CLR      ADC_CLOCK       : CLOCK ON FALLING EDGE
            NOP
            MDV      C.ADC_DATA      ; PICK UP DATA BIT
            RLC      A               ; POSITION
            SETB     ADC_CLOCK
            NOP
            DJNZ     R1,L?RA2        ; LOOP UNTIL DONE

            SETB     ADC_CS          ; DEASSERT SELECT
            RET
            ENDPROC

; ADDRESS TO BIT PATTERN TRANSLATION
.
; INPUT: ACC CONTAINS ADC CHANNEL
 : OUTPUT: ACC CONTAINS 5 BIT ADC CHANNEL SELECT PATTERN
; SINGLE-ENDED OPERATION IS SELECTED. START BIT IS IN MSB
.
XLATE       PROC
            INC      A
            MOVC     A.@A+PC
            RET
            DB       11000000B
            DB       11100000B
            DB       11001000B
            DB       11101000B
            DB       11010000B
            DB       11110000B
            DB       11011000B
            DB       11111000B
            ENDPROC
            END
```

The code in Listing 1 illustrates how to set up the MC146818A for an alarm interrupt based on the current logging interval. Although the MC 1468 18A does well in certain areas, it is not without problems. Let me describe some of the attributes of the MC146818A that I find particularly annoying.

The MC 1468 18A's on-board oscillator is designed to work with crystals up to 4.193 MHz. The time base plays an important role in the accuracy of the timekeeping and the power consumption of the RTC. Quiescent power dissipation of CMOS circuitry consists normally of only leakage currents across reverse-biased

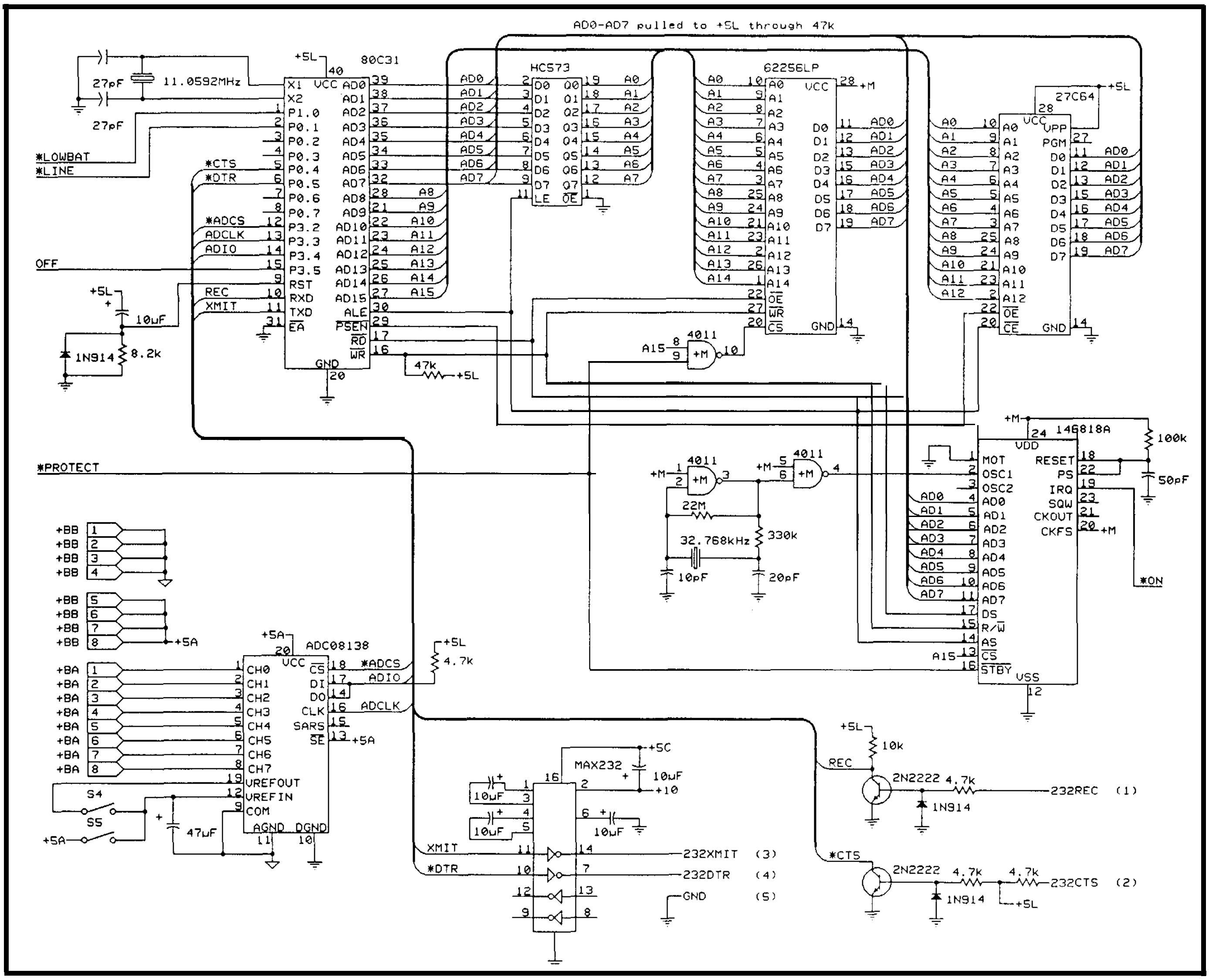


Figure 1—*The 80C31 data logger* ***requires*** *a latch,* ***external*** *memory, and an* ***external*** *reset circuit for tie core. The ADC,* ***RTC,*** *and* ***RS-232*** *interface are added to* ***that.***

diode junctions. Dynamic dissipation consists of two factors associated with switching: the current delivered to the load (primarily the load capacitance] and the current that flows between the supplies during switching when both transistors are momentarily partially on. Both the quiescent and dynamic dissipation are greater at higher voltages. Therefore, the lowest power consumption is attained by running the RTC with a 32-kHz crystal at the lowest possible voltage. Unfortunately, the on-chip oscillator, being sized to operate at up to 4 MHz, draws a lot of current even when operated with a 32-kHz crystal.

The addition of the parallel crystal oscillator composed of two gates from a CD401 1 fixes this problem-to an extent. With this setup, the standby current of the RTC is about 60 µA at 5 volts and drops to 25 µA at 3 volts. This result isn't great, but it is about half of what would be consumed using the RTC's internal oscillator. The second problem with the RTC, especially when used with the 80C3 1 controller, is the way the standby mode is entered. Standby mode is activated by pulling the ● STBY pin low. The wrinkle is that this signal is latched internally by the occurrence of *RD going low followed by ALE going high. Normally this arrangement isn't a problem but remember that all program fetches performed by the 8031 use ‘PSEN, not *RD. A little software monkey business can overcome this deficiency.

I know some of you are thinking I should be using the DS1287 real-time clock module instead of going through all this trouble with the MC146818A. The DS 1287, originally produced by Dallas Semiconductor and now available from a number of manufacturers including Motorola, contains a built-in crystal, lo-year lithium battery, and standby control circuitry, and is almost completely compatible with the MC146818A. I am aware of the merits of this part and have successfully used it in many designs. The bad news is the output pins, including ● IRQ, are totally inoperative when the RTC is running in standby mode, which is true of all the DS1287-type parts that I have seen. If you study the DS1287 further, you will find it has an AC parameter called T_{rec}, or power-on recovery time. Intended to allow the system to stabilize after power on, this recovery period is the time that ● CS must remain high (and the DS 128 7 inaccessible). It is specified as 200 ms maximum and would require a long idle spin, burning all kinds of power waiting for the RTC to come on-line. Definitely unattractive in a battery-powered application.

THE ADC

The ADC used in the data loggers is the National Semiconductor ADC08138 8-bit device. This converter is the improved version of the older ADC0838 and includes an on-chip 2.5volt band-gap-derived reference and track-and-hold front end. If you need more resolution, you can use the ADC1038 that resolves 10 bits, but you'll have to provide your own voltage reference and make a few hardware and software tweaks.

The ADCO8138 is a serial device that communicates with the controller over the Microwire bus, which is essentially a four-wire bus containing transmit, receive, clock, and select pins. By tying the receive and transmit lines together, you can get by using three processor pins for the interface because the receive pin is only seen by the ADC during the addressing interval while the transmit pin is still in a high-impedance state.

This ADC has a flexible multiplexing scheme that allows operation as an eight-channel single-ended converter referenced to ground, an eight-channel pseudo differential converter referenced to a common pin that may be biased to some arbitrary voltage, or as a four-channel differential mode converter. When used differentially, the polarity for the differential inputs can also be selected. Actually, any combination of modes is available because the front end is configured each time a conversion is started as part of the addressing sequence.

A/D conversion is performed using the standard successive approximation method referenced to the data clock. National Semiconductor recommends holding a 40% to 60% duty cycle on the clock, which is easily accomplished by sprinkling a few NOPs through the ADC code. The ADC code for single-ended operation is shown in Listing 2.

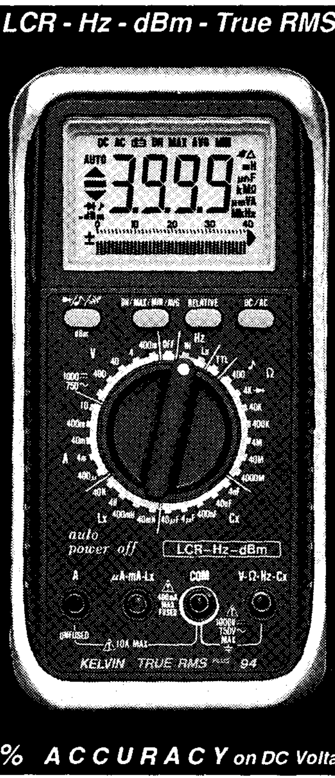
RS-232 INTERFACE

Communications with the data loggers are carried out in accordance with the RS-232 standard, with the level shifting on the outgoing signals performed using a MAX233 interface chip. Similar to the MAX232, this 20-pin IC implements its internal charge pump functions without the need for any external capacitors. Vcc to the MAX233 is switched using a PNP transistor turned on only in the presence of line power in order to conserve power when running off the battery. As you may have discovered, with the MAX233 powered off, the application of an RS-232 signal to the receiver input has the undesirable side effect of partially powering the chip. Furthermore, this voltage has a tendency of backfeeding the power supply pin, which biases the system Vcc rail to an indeterminate level if the main power is off. Although not harmful in itself, this phenomenon may result in excessive battery drain and the possible malfunction of the RTC or RAM protection circuits, resulting in the corruption of data.

Not feeling lucky, I elected to avoid this problem altogether by using an NPN transistor as a receiver. A 4.7 kΩ series resistor limits the line current and a 1N914 signal diode with the cathode wired to the base and its anode to ground limits the negative excursions at the base to a safe level. Providing a pull-up resistor on the transistor's collector is also a good idea because the rather strange three-transistor arrangement used on the CMOS 80C3 1 I/O pins makes for the very slow rise times I have occasionally observed, which may disrupt the operation of the 80C3 l's on-chip UART.

Actually, the transistor receiver is not all that bad. The MAX chips do incorporate hysteresis on their receivers, but they really don't make use of the negative signal component. That is to say, the MAX233 (and other Maxim RS-232 chips as well) will operate just fine on an input signal with a 0-5-volt range.

LCR - Hz - dBm - True RMS



1 % ACCURACY on DC Voltages

TRUE RMS PLUS Model 94

MAX/MIN/AVERAGE MEMORY RECORD

RELATIVE MODE / DATA HOLD

DC/AC VOLTMETERS
DC Range: 400mV. 4V,40V,400V,1000V
A C Range: 400mV, 4V, 40V, 400V, 750V

DC/AC AMP METERS
DC/AC Ranges: 400uA, 4mA. 40mA, 4OOmA. 10A

OHM METER
Range: 400, 4K, 40K. 4OOK. 4M, 40M, 4000M Ohms

FREQUENCYCOUNTER-AUTOWINGING
Range: 4KHz, 40KHz, 400KHz, 4MHz (Trigger Low).
20MHz (Trigger High)

LOGIC PROBE

AUDIBLE CONTINUITY TESTER

CAPACITANCE TESTER
Range: 4nF, 40nF, 400nF, 4uF, 40uF

DIODE TESTER

INTRODUCTORY OFFER!

dBm TESTER
Range: -25.7 dBm to 59.7 dBm

INDUCTANCE TESTER
Range: 40mH. 400mH, 4H, 40H

10 MEGA OHM IMPEDANCE

10A HIGH-ENERGY FUSE PROTECTION

AUTO SLEEP & AUTO POWER OFF

WATER & DUST RESISTANT

MODEL 94 cOMeS COMPLETE WITH
TEST LEADS, YELLOW HOLSTER,
TILT STAND, BATTERY & FUSE

Stock No.
990111
LIST PRICE $295

ONLY $199.95

2 YEAR WARRANTY

KELVIN

10 HUB DRIVE, MELVILLE, NY 11747
(800) 6459212
(516) 756-1750 (516) 756-1763/FAX

138

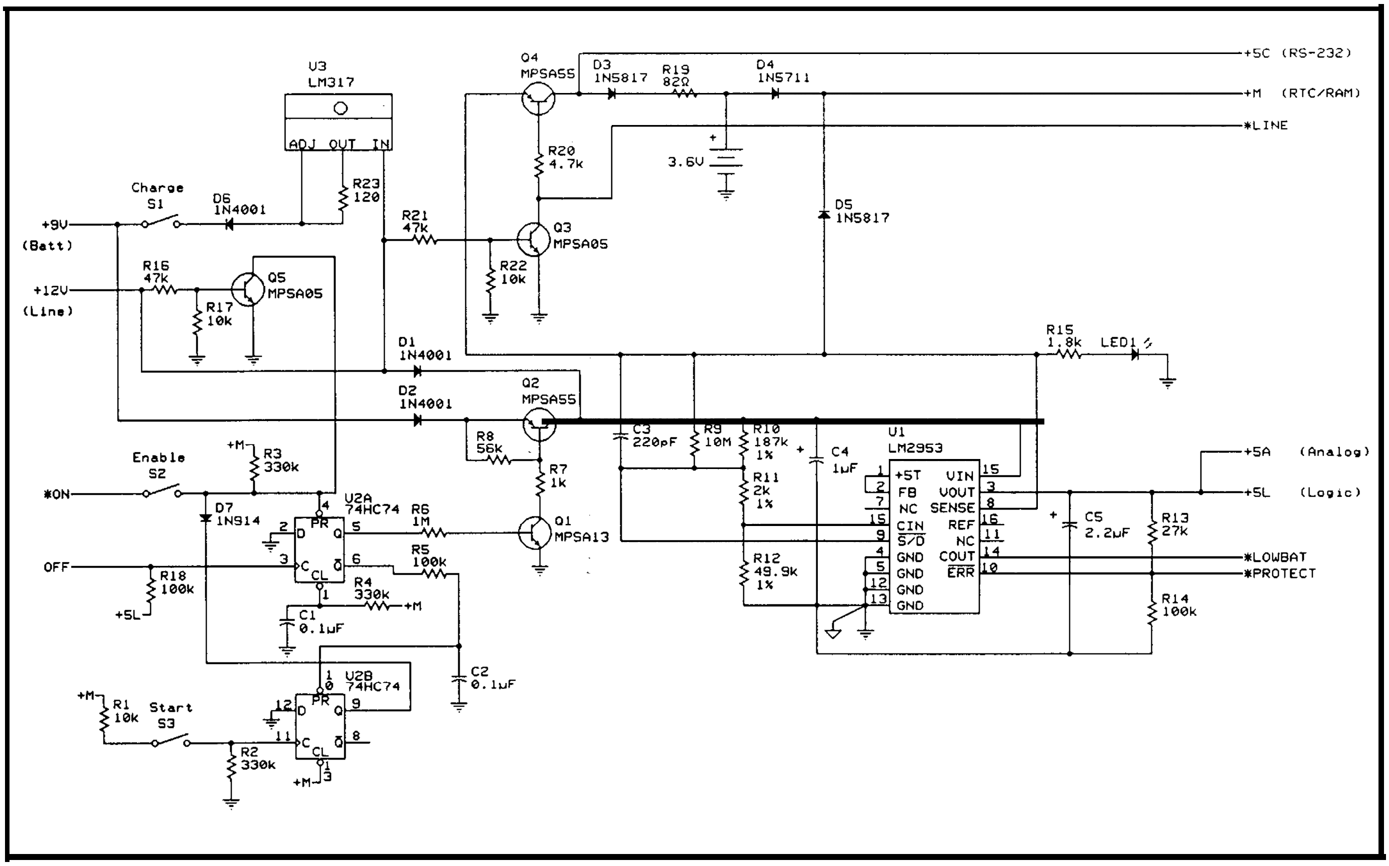
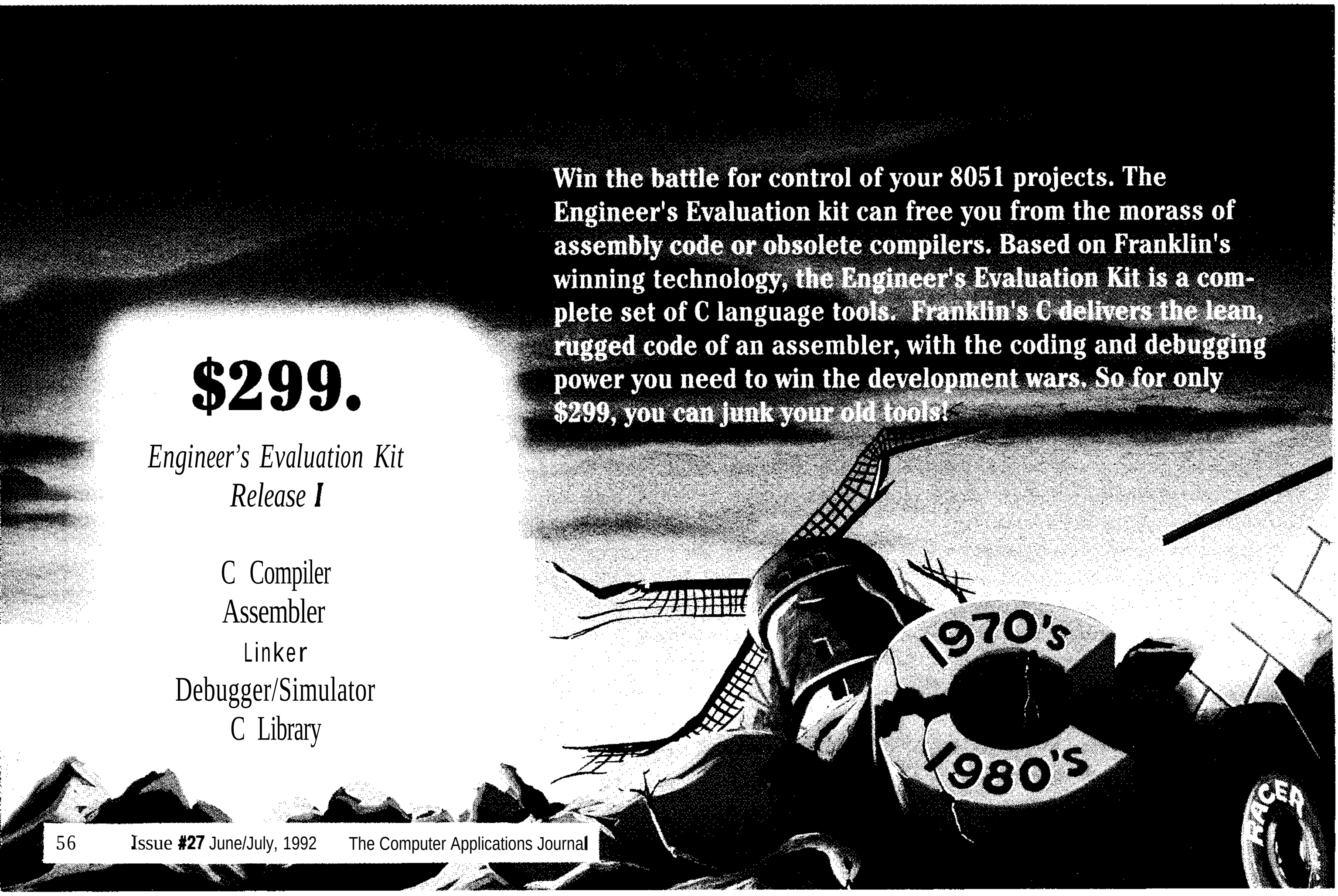


Figure 2-The *power* manager *circuit* for *the 80C31 logger provides* separate *power supplies for* the *logic sections (+5L),* the analog section *(+5A),* memory *(+M),* and *communications (+5C).*


Win the battle for control of your 8051 projects. The Engineer's Evaluation kit can free you from the morass of assembly code or obsolete compilers. Based on Franklin's winning technology, the Engineer's Evaluation Kit is a complete set of C language tools. Franklin's C delivers the lean, rugged code of an assembler, with the coding and debugging power you need to win the development wars. So for only $299, you can junk your old tools!

$299.

Engineer's Evaluation Kit
Release I

C Compiler
Assembler
Linker
Debugger/Simulator
C Library

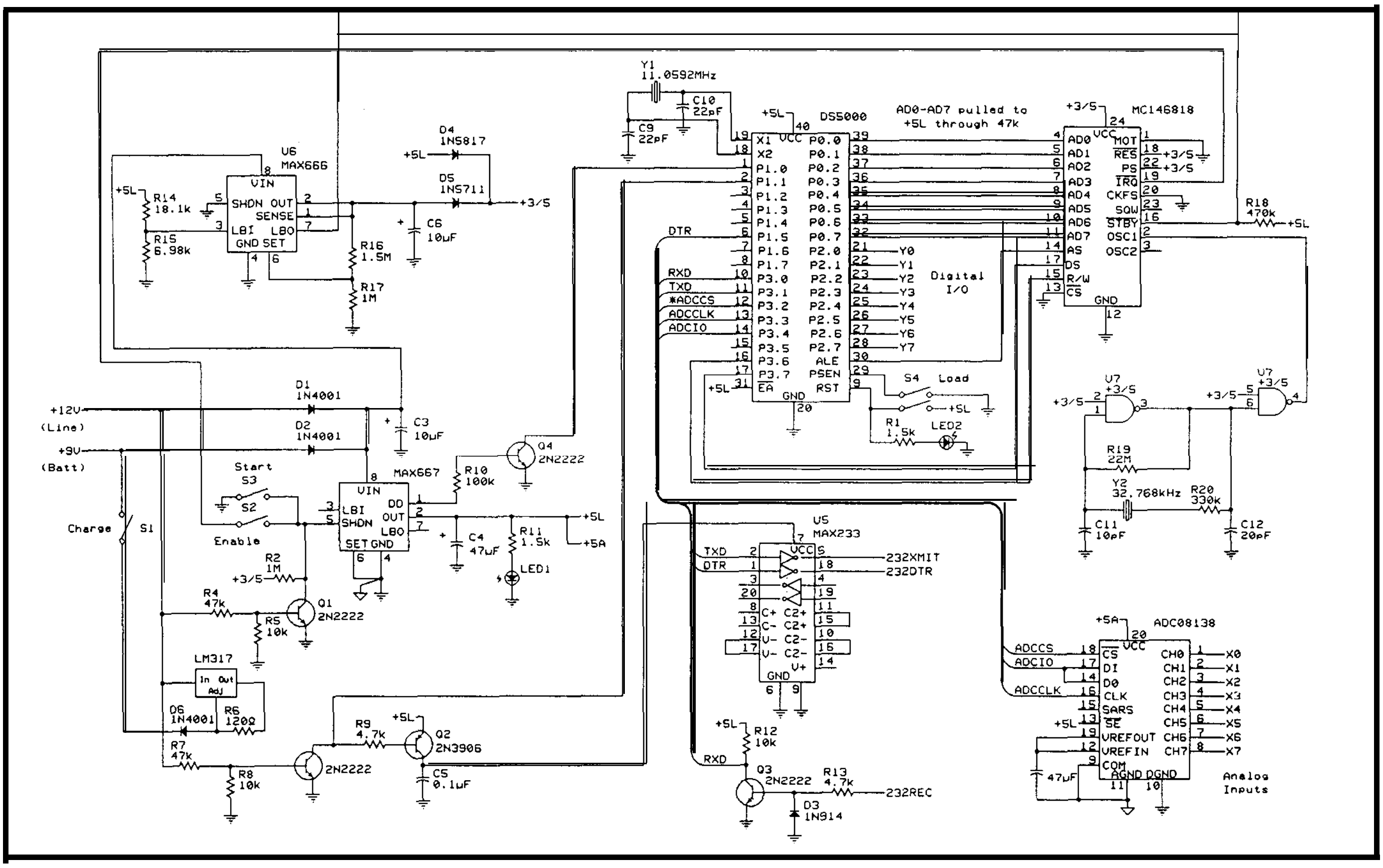
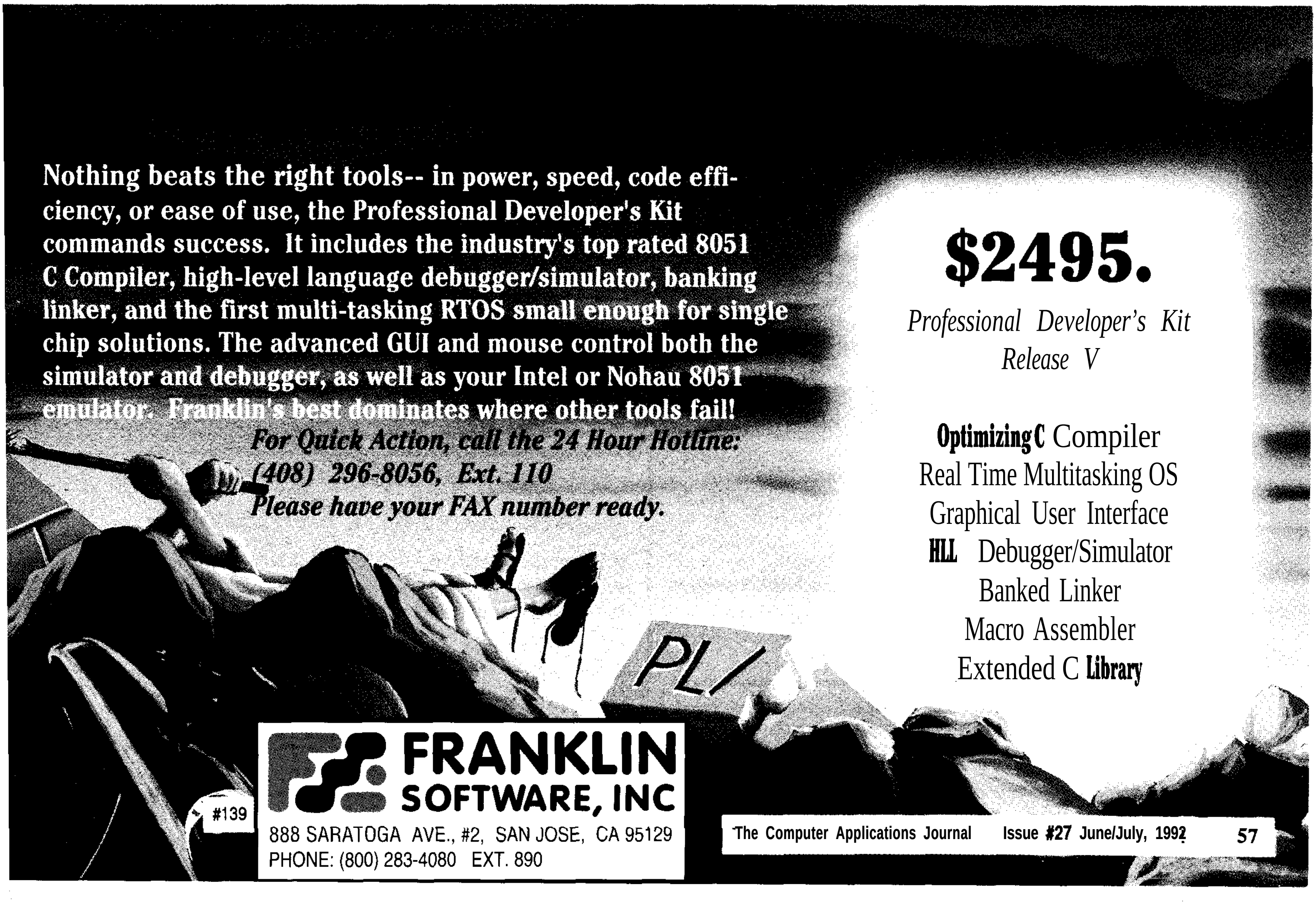


Figure 3—*Redesigning* the data *logger* with a *DS5000 eliminates* much of *the processor support* circuitry and simplifies the *power management section.*


Nothing beats the right tools-- in power, speed, code efficiency, or ease of use, the Professional Developer's Kit commands success. It includes the industry's top rated 8051 C Compiler, high-level language debugger/simulator, banking linker, and the first multi-tasking RTOS small enough for single chip solutions. The advanced GUI and mouse control both the simulator and debugger, as well as your Intel or Nohau 8051 emulator. Franklin's best dominates where other tools fail!

For Quick Action, call the 24 Hour Hotline:
(408) 296-8056, Ext. 110
Please have your FAX number ready.

$2495.

Professional Developer's Kit
Release V

Optimizing C Compiler
Real Time Multitasking OS
Graphical User Interface
HLL Debugger/Simulator
Banked Linker
Macro Assembler
Extended C **Library**

#139

FRANKLIN
SOFTWARE, INC
888 SARATOGA AVE., #2, SAN JOSE, CA 95129
PHONE: (800) 283-4080 EXT. 890

THE GOOD, THE BAD, AND THE UGLY

As I mentioned, I designed two different data loggers. Both are 80C3 1 based, although how I obtained the 80C3 1 functionality differs quite a bit. In one case, I used the Intel 80C3 1 controller, an address latch, reset circuit, PROM, RAM, discrete RAM protection circuitry, and a RAM backup power source. For the other device, I was able to replace all these components with a Dallas DS5000. The only other difference is in the power supply sections and the backup power source.

You can judge for yourself what's good, bad, or ugly. I can say that both versions work reliably, but I know which one I prefer. Let me describe the 80C3 1 -based logger first.

THE 80C31 LOGGER

In the interest of keeping this epic within manageable proportions, I will omit a description of the 80C3 1 controller section. This singularly uninteresting configuration consists of the usual 80C3 1, PROM, RAM, and glue as previously mentioned and should be familiar [see Figure 1).

Figure 2 depicts the 803 1 data logger's power supply section. Battery and line power are combined using mixing diodes Dl and D2, and this voltage is presented to the +5-volt regulator Ul, a National Semiconductor LM2953. The LM2953 is a low-dropout regulator featuring a moderately low quiescent current of about 1 mA at a 50-mA load and very tight output regulation. Also included on this chip is a comparator that I use as a low-battery monitor set up to trip at about 5.8 volts.

A built-in crowbar is incorporated into the regulator, which can be wired to pull the output down quickly when the shutdown pin is activated. The use of the crowbar is intended to prevent operation with an indeterminate voltage and forces the regulator to function in a snap-on/snap-off mode at voltage levels of 5.8 volts and 5.7 volts, respectively. The *ERR output pulls low when the regulator starts losing regulation and is wired to the MC146818A's*STB pin and to the RAM's enable gate U4. The other function of U4 is to locate the RAM at 8000h by virtue of its inverting action. R14, the 100 kΩ pull-down resistor on the 'ERR pin, ensures this line is at a low level when the regulator ceases operating.

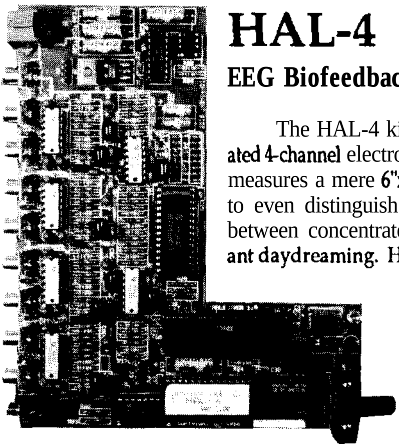
At first, my natural inclination was simply to use one of the integrated RAM backup circuits, such as the Dallas DS1210. However, aside from its rather long recovery-time parameter, it may have caused other problems as well. Let me explain the problem before I proceed.

The DS1210 enters standby mode when Vcc drops below a selectable trip point, at which time the chip select line to the RAM is clamped high. This point can be set to a nominal 4.62 volts or 4.37 volts (the corresponding minimums are specified at 4.25 volts and 4.5 volts). The problem is that the DS1210 doesn't actually monitor the Vcc pin as you might expect. What is actually sensed is the internal chip

CIRCUIT CELLAR KITS

HAL-4

EEG Biofeedback Brainwave Analyzer



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

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

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

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

Sonar Ranging Experimenter's Kit

Targeting ◆ Ranging ◆ Machine Vision

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

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

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

CHECK OUT THE NEW CIRCUIT CELLAR
HOME CONTROL SYSTEM

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

Call and ask about the *HCS II*

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

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

#14

voltage obtained by summing the battery and Vcc pins. With the three-cell NiCd battery I am using, I can expect a float voltage of about 4.35 volts, which means the DS1210 may never go into standby mode (because the battery voltage may exceed the Vcc trip point specification). Dallas plays it safe and lists the maximum battery level at 4 volts. Obviously this part is meant to be used with a lithium cell! The important point here is that understanding how this chip works will keep you out of trouble. I have used it for years with a supercap power source (with charge voltage limiting) and have recently implemented a two-cell NiCd backup scheme for a large memory array with no problems whatsoever.

Backup power to the RTC and RAM is derived from a 3.6-volt NiCd battery composed of three 60-mAH cells. This battery receives charging current from a semiconstant current source made up of the blocking diode D3 and limiting resistor R19. Because this battery will attain a float voltage of 4.35 volts, the blocking diode is a Schottky type in order to minimize the drop, ensuring an adequate charging potential from the +5-volt source. Mixing diodes D4 and D5 route uninterruptable power to the RTC and RAM circuits, as well as to the power control flip-flops.

The charger for the main battery consists of an LM3 17 (U3), R23, and D6, wired as a constant current source set to deliver the 10 mA charging current. At this rate, the recharge time is 16 to 20 hours, at which point the cells reach a maximum potential of about 1.45 volts (10.15 volts for the battery). The series blocking diode prevents backfeeding the LM3 17. Switch S1 disables charging in case you want to use a nonrechargeable battery and don't want to blow up the data logger.

Incoming line power is detected up by transistor Q3. The collector of this transistor is monitored by the controller to determine if it is running off of line power. The collector also connects to Q4—the +5-volt PNP switch that goes on when line power is present, providing power to the communications circuit and charging current to the backup battery.

The main battery power is controlled by a D-type flip-flop U2a that accepts inputs from the RTC's ● IRQ pin and from a manual push-button switch. When an alarm condition occurs, the MC146-818A asserts its 'IRQ signal. If the disable switch is closed, *SET is driven to its active state and forces the Q output high. This action results in the application of base bias to the Darlington transistor Q 1, which saturates the main PNP battery switch Q2, providing power to the system. Being connected to an 803 1 port pin, the OFF signal that ties to the flop's CLK input goes high at this time, which is the I/O pin's default state.

Remember that on a D flip-flop, the synchronous inputs-•'CLR and *SET-take precedence over the asynchronous CLK input, so CLK is not seen by the flip-flop at this time. Once the controller regains control, the RTC's alarm condition is cleared, thereby releasing ● IRQ and effectively enabling the CLK input. The controller is now able to power down the system at any time by issuing a rising edge on CLK.

U2b is used as a pulse generator for the manual push-button input. The output of this section connects to the *SET input of the master flip-flop through a diode that, in essence, transforms the Q pin into an open-collector output. Finally, R4 and Cl ensure the circuit assumes the off state upon initial application of power.

THE DS5000 LOGGER

Take the 80C31, add some RAM, PROM, a nonvolatile RAM controller, reset circuit, along with some glue and you have the logic element for our data logger. The Dallas DS5000 is composed of all of these components and includes the lithium RAM backup battery and many other features as well. Housed in a 40-pin DIP package, the DS5000 not only provides an industry-standard footprint but also retains use of all of the ports for general-purpose I/O. (All embedded memory accesses are carried out over a separate internal bus.)

REALIZE
THE
POWER
OF...

[illegible]

Call for Information
Toll-Free 1-800-537-5043

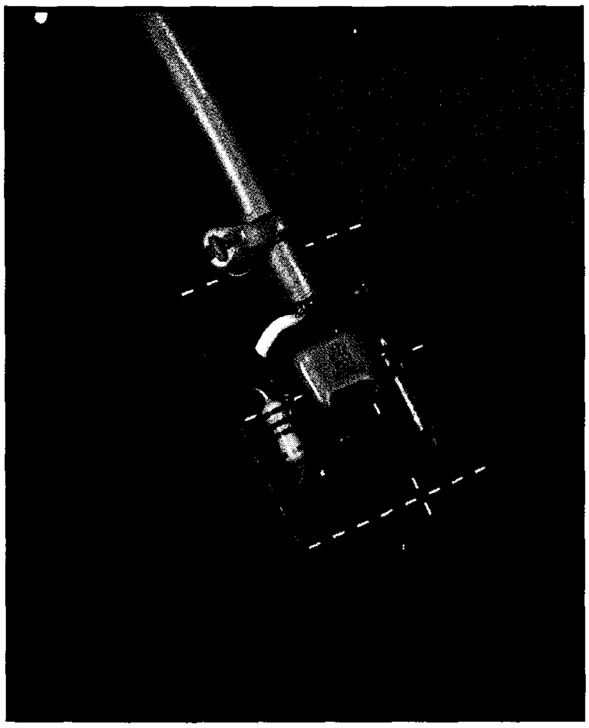
PARADIGM
The Model for Programming Productivity

3301 Country Club Road
Suite 2214
Endwell, NY 13760
(607) 748-5966
FAX: (607) 748-5968

All trademarks are property of their respective holders.

Referred to by some as a Cadillac in the Volkswagon market, the DS5000 is admittedly pricey. As such, it does not compete for most 803 1 sockets. Of course, looking at things in perspective, have you priced 87C51s lately? Actually, due to the high integration the DS5000 possesses, it contains many of the elements of the lower-end microcontroller boards on the market and it may be more fair to justify its cost in that context. I rather like the idea of having reduced the entire controller section to a single component, both on paper and on the board!

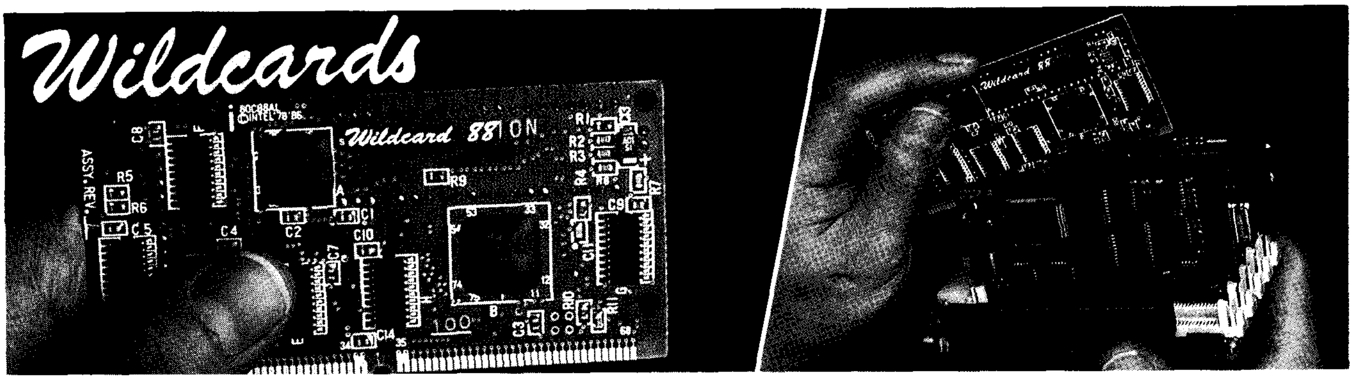
One of the more powerful features of the DS5000 is its onboard bootstrap loader. All nonvolatile areas within the DS5000 may be completely reloaded by invoking the serial loader and issuing the appropriate commands via the serial port. This mode is entered by pulling RST high and 'PSEN low and is accomplished on the logger with a DPST switch, S4, wired appropriately. I include an LED to indicate when this mode is in effect.



A simple **temperature sensor using the popular LM34 may be connected** *to either da& logger.*

A number of useful functions can be accessed in bootstrap mode, such as dumping the CRC- 16 checksum of embedded RAM, filling embedded RAM with a specified byte, dumping embedded RAM in Intel hex format, verifying embedded RAM with an incoming Intel hex file, and, of course, loading an Intel hex file. The MCON register may also be written to in this mode, permitting the setting of the partition address (the address where program memory ends and data memory begins). Combined with a PC-resident loader utility, the DS5000 can serve as a mini embedded system development platform, obviating the need for a ROM emulator to develop simple applications. (For more information on the DS5000, see my article in issue #16 of *Circuit Cellar INK,* “ONDI-The ON-line Device Interface,” describing the ONDI remote

2”x 4” EMBEDDED PC



Microcontroller. Microcomputer.

''*Megatel Wildcards* provide PC *functionality* in a *flexible, small* format.”

Wildcard 88’”
- CPU **clock to** 10 **MHz**
- **Replaces full PC motherboard**
- **Co-processor and BIOS socket**
- **DMA. Bus, DRAM, Keyboard controllers**

Multi/10
- On-board SCSI Host Adapter **(supports up to** 7 **devices)**
- **Floppy Controller** (1.44M,1.2M)
- **2 RS-232,** 1 **Parallel.** 1 **RS-485 multi-protocol serial port**

Vid/Mem:
- **640Kb User memory**
- Video/Colour **LCD controls CGA, Hercules@, IBM@ Mono: (runs LCD Panels)**

For information on our *representatives please* contact our head *office at* the number below.

(416) 245-3324

All Wildcards are low power single +5 volt operation.

125 **Wendell Ave., Weston, Ont.** M9N3K9 **Fax: (416)** 245-6505

Wildcard 88 and Megatel are trademarks of Megatel Computer Corp Hercules is a trademark of Hercules Corp IBM is a trademark of IBM Corp.

megatel®

#142

control that featured the functionally equivalent DS2250.)

The DS5000 is available with either 8K or 32K of embedded lithium-backed RAM, with or without an integrated battery-backed real-time clock, and with speed options of 8, 12, or 16 MHz. For the data logger, I picked the 12-MHz part with 32K of embedded RAM and no RTC.

Containing an internal power-on reset capability, the DS5000 requires no external components to perform this function. The reset signal is generated by the internal control circuitry to allow the oscillator to start up from its halted state, which is in effect when Vcc is below 4.5 volts. The delay circuit counts 21,504 oscillator periods (1.9 ms at 11.0592 MHz) before releasing the internal reset line. The sooner you can come up and take care of business the less power will be consumed. Actually, the system comes up fast enough to necessitate the insertion of a small delay, about 5 ms, prior to entering the data collection code. I found that the logger was ready to roll before the capacitor on the ADC's reference pin had a chance to ramp up, which resulted in errant data conversions.

As shown in Figure 3, when used with the DS5000, the MC 1468 18A's on-board address latch eliminates the need for any glue. Also, note that all the pins of port 2 are available for use as general I/O because the RTC is referenced using MOV X A, @R0–type instructions that have no effect on the high-order address bus.

The MAX667 low-dropout pass regulator (U4) steps down the main battery power to 5 volts providing power to the controller, ADC, and support circuitry. This regulator features very low quiescent current—on the order of 20 µA—and a typical input/output differential of 150 mV at 200 mA. Pulling the SHDN pin up to a level over 1.5 volts disables the output power and brings the regulator's quiescent current down around 0.2 µA.

The SHDN pin is controlled directly by the RTC's ● IRQ line and also connects to the collector of NPN transistor Q1 that forces the regulator on in the presence of line power. The omission of the power control flip-flop in this circuit does not really have anything to do with the regulator itself. After gaining a better understanding of how the MC146818A worked, I discovered that it was not required after all! (I'm hopeful that with time we get smarter.) This omission is possible because when indicating an alarm interrupt, the MC146818A forces 'IRQ low for the duration of time the alarm interrupt bit in register C is set. After performing its data acquisition task, the controller reads the RTC's C register, clearing the alarm interrupt bit and releasing ● IRQ, thereby disabling power to the system.

DD is the dropout detector pin, which is the collector of a PNP transistor that changes as the dropout voltage approaches its limit. DD starts to source current when the regulator begins to lose regulation and is dependent on the input voltage, output current, and temperature (as is the dropout voltage). Connected to an NPN transistor Q4, this signal generates the low-battery indicator to the controller.

Rather than include a separate battery for the RTC, I decided to add a

Simple. Elegant. Reliable.

Features

❒ Priority based scheduler with time slicing and preemption options
❒ Unlimited number of tasks, queues, resources, and events
❒ Fixed and Variable memory management
❒ Time-out option on all service calls
❒ ROMable C source code included
❒ No royalties
❒ MS-DOS compatible file system

NUCLEUS RTX
Real-time Multi-tasking Executive

Simple solutions are generally better solutions. Complicated solutions often reflect poor design. Nucleus embodies simplicity. Its elegant style and design make it easy to use and understand. Nucleus is the best, most reliable choice for your embedded application.

Processors Supported

80x86, 80386/486 protected mode, 68xxxs, 683xxs, Am29000 RISC family, Intel's i960, SPARClite, MIPS, TMS320C3x and the TMS320C2x.

Let us prove it.
Call now for more information.

(800) 468-NUKE
ACCELERATED TECHNOLOGY INC
P.O. Box 850245 Mobile, AL 36685
(205) 450-0707

#146

second regulator to provide the uninterruptable power for the timing subsystem. The MAX666(U6), with its typical 6-µA quiescent current and 40-mA output capability, does the job nicely. LBI, the low-battery input, monitors the logic +5 volts and trips the output, LBO, at 4.75 volts on the way down, providing the ● STBY signal to the MC146818A. I was able to lower the RTC's standby current by operating the '666 at 3.25 volts after biasing the SET pin with a couple of trim resistors, R16 and R17. Finally, the 3.25volt feed is summed with the 5 volts using two Schottky diodes, D4 and D5, in order to provide the proper operating voltage to the RTC when the main regulator is active.

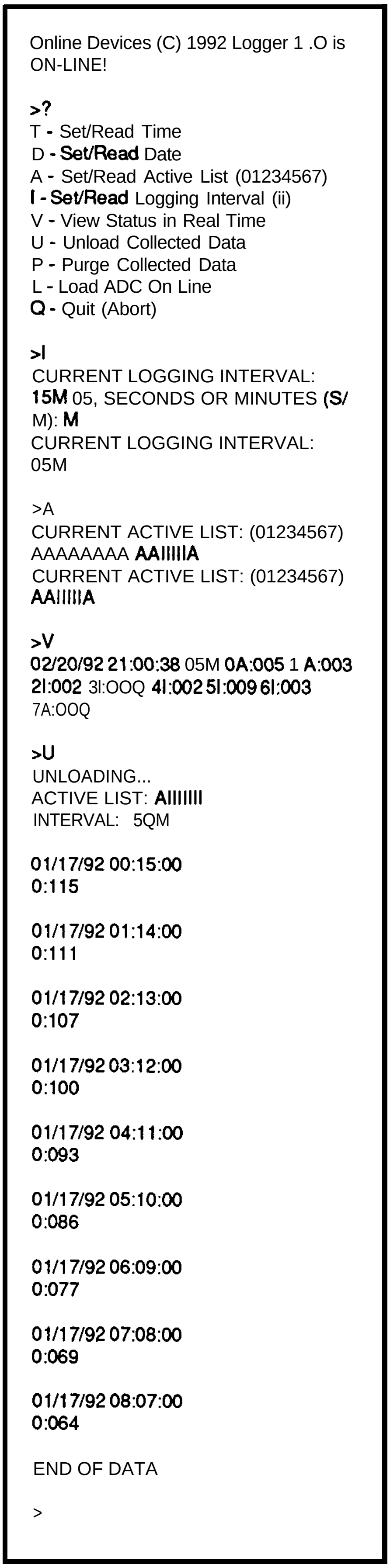
CODE

The data logger control program runs about 4K in size and is coded entirely in assembly language. Fundamentally, this program is composed of three parts: the boot code, the console code, and the data collection code.

The mainline gets control immediately after the boot sequence completes and simply decides whether to proceed to either the console code or the data collection code. This decision is made on the state of the *LINE pin. When *LINE is low, indicating that the logger is operating under line power, control passes to the console code. A high level on *LINE indicates the data logger is running under battery power and, in this case, the main data collection code is executed.

Although the data logger is downloadable, and to an extent programmable, this aspect is primarily intended to provide the system with different personalities for various data collection tasks. A facility is required to configure and check out the system prior to performing any data logging, which is accomplished from the console mode.

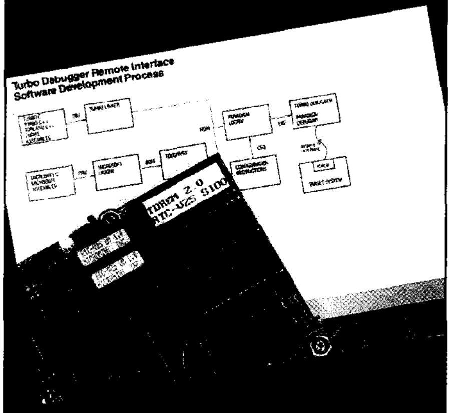
On entry into console mode, the data logger presents the log-on message shown at the top of Figure 4. At this point, any of the supported commands may be executed directly, or a list of choices may be viewed by pressing the "?" key.



```
Online Devices (C) 1992 Logger 1 .O is
ON-LINE!

>?
T - Set/Read Time
D - Set/Read Date
A - Set/Read Active List (01234567)
I - Set/Read Logging Interval (ii)
V - View Status in Real Time
U - Unload Collected Data
P - Purge Collected Data
L - Load ADC On Line
Q - Quit (Abort)

>I
CURRENT LOGGING INTERVAL:
15M 05, SECONDS OR MINUTES (S/
M): M
CURRENT LOGGING INTERVAL:
05M

>A
CURRENT ACTIVE LIST: (01234567)
AAAAAAAA AAIIIIIA
CURRENT ACTIVE LIST: (01234567)
AAIIIIIA

>V
02/20/92 21:00:38 05M 0A:005 1 A:003
2I:002 3I:OOQ 4I:002 5I:009 6I:003
7A:OOQ

>U
UNLOADING...
ACTIVE LIST: AIIIIIII
INTERVAL:  5QM

01/17/92 00:15:00
0:115

01/17/92 01:14:00
0:111

01/17/92 02:13:00
0:107

01/17/92 03:12:00
0:100

01/17/92 04:11:00
0:093

01/17/92 05:10:00
0:086

01/17/92 06:09:00
0:077

01/17/92 07:08:00
0:069

01/17/92 08:07:00
0:064

END OF DATA

>
```

Figure 4-A *test* ***run*** of the data ***logger chronicles*** he output of *a* ***wood*** stove *on a cold winter night.*

REALIZE THE POWER OF...



[illegible] ultimate flexibility.

Call for Information!
Toll-Free 1-800-537-5043

PARADIGM

The Model for
Programming Productivity

3301 Country Club Road
Suite 2214
Endwell, NY 13760
(607) 748-5966
FAX: (607) 748-5968

All trademarks are property of their respective holders.

The conventions I adopted are simple and work in a consistent manner. When a function is invoked, if dealing with a parameter that can be modified, the current value of the parameter is first displayed and a prompt is issued for a new entry. At this point, you may enter a new value, or abort the function by hitting the "Q" key.

Obviously, the time and date functions read or set the real-time clock. Using the "A" and "I" commands, the active list and interval time may be read or set, defining the analog points that will be logged to memory and the amount of time the logger will wait between readings. "V"iew displays the date and time, logging interval, and the eight analog inputs in real time along with their status as active or inactive.

"U"nload performs a data dump to the computer for displaying or logging to disk. When simply displaying the collected data, the space key may be used to pause the dump, and striking any key resumes activity. The dump of a test run, shown at the bottom of Figure 4, chronicles the output of my wood stove on a cold winter night.

"P"urge clears the data buffer of all collected data (the active list and interval time are unaffected).

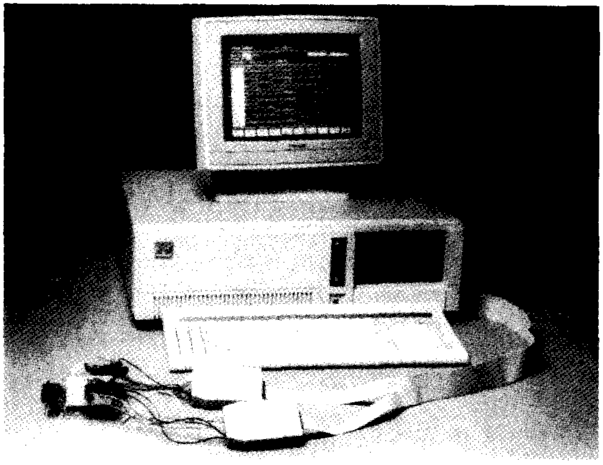
Finally, "L"oad allows doing an on-line load to memory in accordance with the active list and interval time.

All communications are performed at 9600 bps, with SIO synchronization performed by polling the UART. Although breaking old habits is hard, and I've generally implemented my communications routines on an interrupt-driven basis, polling is more appropriate for the logger's communications needs because things generally operate on a stop-and-wait basis. The screen formatting is intended for a computer running a communications program operating as an ANSI terminal. Where there is a lot of output, such as in the Unload and View modes, XON/XOFF protocol is supported in order to prevent swamping the computer, which is especially important when logging data to disk.

Upon removal of line power, the logger goes into its idle state. At this time, the RTC interval timer is disabled. The push-button switch may now be used to activate the logger, and if any of the analog points are enabled via the active list, a conversion is performed and stored. Furthermore, if the interval time is set to a nonzero value, the interval timer is started prior to a self power down, and data collection now proceeds automatically at the selected interval. Data collection can be suspended by opening the disenable switch or entering console mode by attaching line power. Once in console mode, it is possible to modify certain parameters, such as the active list or interval time, before resuming data collection without affecting data already stored. Certain fault conditions, such as a low battery or full buffer, will terminate data collection by killing the RTC interval timer and powering down the system in order to safeguard any data already in RAM.

To conserve memory, certain items are stored only when they

PC-Based Logic Analyzers



New Prices for 1992
ID160 (50MHz) $595
ID161 (100 MHz) $695
*High Speed •8K Trace Buffer • 16 timing channels expandable to 32 state channels *Multi-Level Triggering *State Pass Counting • Event Timer/Counter *Performance Histograms *Hardcopy Output • Disassembles 8-bit micros *Supports VGA /EGA/HGA • Demo diskette available
30 Day Money Back Guarantee

INNOTEC DESIGN, INC.
6910 Oslo Circle, Suite 207
Buena Park, CA 90621
Tel: 714-522-1469 FAX:714-527-1812

#150

BACKGROUND MODE EMULATOR
68332
68340
68HC16
and 68330, 68331, 68333
Complete hardware testing and C source level debugging for thousands less than an ICE.
Call for a free trial unit.

(617) 828-5588
EST
Embedded Support Tools Corp.
10 Elmwood St. Canton, MA 02021

#151

change. These include the date and time, the logging interval, and the active list. Analog conversions are stored in raw binary format and consume one byte per channel. When unloading the data to a PC, the various elements, such as the date and time, are picked out of the storage buffer and reconstructed into a meaningful format. The active list, being stored each time it is modified, is used by the extraction routine to determine how many analog readings there are per data record and allows tagging these readings with their appropriate channel IDs at transmission time.

HOW LONG IS LONG?

The 80C3 1 data logger uses separate batteries for the RAM and RTC power, and the logic elements and converter. As I mentioned earlier, the current consumption of the real-time clock circuit varies depending on the level of Vcc. This amount can vary from about 60 µA at 4.35 volts to 25 µA at 3 volts because it is being powered directly from the backup battery in standby *mode. If you* split the difference

$$\frac{60\ \text{mAH}}{0.042\ \text{mA}} = 1428.5\ \text{hours}$$

you get a backup time of about two months.

Operating current is consumed at the rate of 35 mA, which is maintained for about 70 ms. Most of the power is wasted because of the less-than-optimal power-on reset circuit; the actual processing time is really quite short. Accepting these values, each conversion consumes

$$\frac{35\ \text{mA} \times 70\ \text{ms}}{3600\ \text{s/hr}} - 0.68\ \mu\text{AH}$$

of battery capacity.

Using the above value, you should be to get

$$\frac{100\ \text{mAH}}{0.68\ \mu\text{AH}} = 147{,}058\ \text{samples}$$

before the NiCd battery goes flat.

Although the power requirements of the timekeeping and processing sections are quite different, the main battery provides power to both the backup and active circuits in the DS5000 data logger.

In standby mode, with current consumption running about 25 µA, the previous calculations indicate the timekeeper will run for over 5 months off of the NiCd battery. Using the alkaline battery, over two years can be attained. The memory is another story. The RAM is powered by the DS5000's internal lithium power source, and Dallas guarantees a minimum data retention time of ten years in the absence of primary power. This support is good because the RAM also contains the program code.

During data acquisition, the DS5000 logger consumes 30 mA. The good news is this rate of consumption is maintained for a very short time. The duration during which actual processing takes place is on the order of 5 ms as in the 80C31 logger. Add another 20 milliseconds or so to allow for the reset interval, for the power-on delay, and for Vcc to ramp up and down, and basically that's it. Figure on 30 ms to be safe and you can calculate the amount of battery capacity used per sample, which is 0.25 µAH.

Considered alone, this time frame indicates that 400,000 samples can be taken using the lower capacity NiCd. Although quite impressive in itself, the number is frankly ridiculous in practical terms. ❑

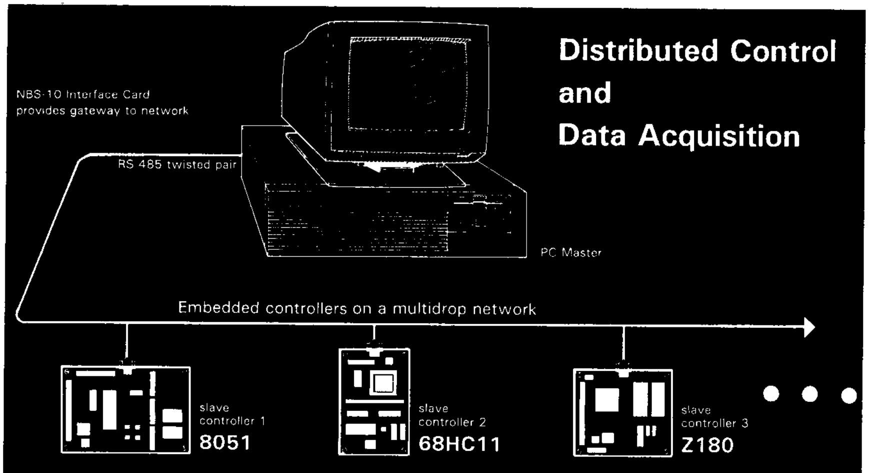
John Dybowski has been invloved in the design and manufacture of hardware and software for industrial data collection and communications equipment.

SOFTWARE

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

I R S

413 Very Useful
414 Moderately Useful
415 Not Useful



Distributed Control and Data Acquisition

Connectivity is implemented at low cost with a network designed by CIMETRICS TECHNOLOGY.

Our hardware/software solution includes:
Microcontroller code, PC/XT/AT interface card, NSP protocol, and application development software toolkit.

CIMETRICS TECHNOLOGY
the network solution
for embedded controllers

120 West State Street, Ithaca, New York, 14850
Phone: 607-273-5715 FAX: 607-273-5712

#152

DEPARTMENTS

68 Firmware Furnace

76 From the Bench

84 Silicon Update

92 Practical Algorithms

99 ConnecTime

An HCS II LCD Terminal

The Circuit Cellar HCS II interfaces well with real-world sensors and actuators, but what about the human interface? Ed shows us a simple LCD terminal that can be added anywhere in the system.

FIRMWARE FURNACE

Ed Nisley

Once you have the HCS II essentials down pat, the WIBNIs (Wouldn't It Be Nice Ifs) appear. Both Steve and Ken thought, "Wouldn't it be nice if there was some way to display HCS II status and debugging information?" After all, putting display terminals right on that RS-485 network with all the other gadgets makes sense, right?

My topic this issue is, truly, a simple matter of firmware: an RS-485 networked LCD terminal called LCD-Link. In addition to character display, the firmware handles ANSI cursor-positioning commands, implements C-style character escape sequences, and remembers which buttons have been pressed until the Supervisory Controller (SC) can interrogate it.

Longtime readers will recall the Furnace Firmware project included an assembly language ANSI LCD driver [see Circuit *Cellar* INK, issue #17). This time around I used Micro-C so you can compare and contrast the two approaches. The earlier code is both smaller and faster, but this version is small enough, fast enough, and a lot easier to understand. Take your pick!

SIMPLE HARDWARE

LCD panels based on the Hitachi HD44780 controller are old friends, having appeared in a variety of my projects since I discussed them in issue #7. Figure 1 shows the requisite circuitry: one such LCD panel, four switches, and the same COMM-Link board we've used for the previous projects in this series (see *Circuit Cellar INK*, issue #25). Can't get much simpler than that!

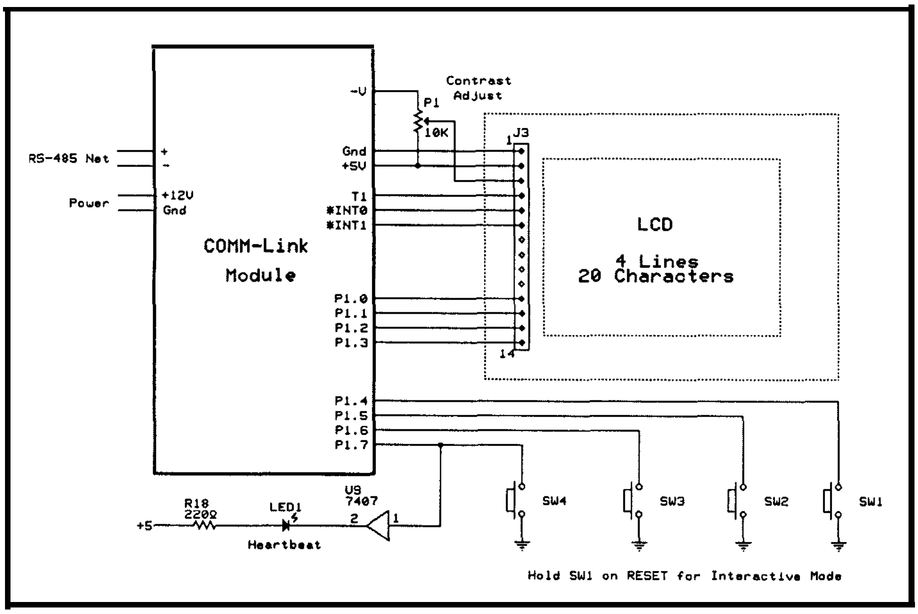


Figure l-The LCD-Link *is* based *on the* same *COMM-Link* module as *most* of *the other HCS II network modules. The addition* **of** *an LCD display,* ***four buttons,*** *and some* ***custom*** *firmware complete the package.*

The HD44780 supports a CPU interface with either 4 or 8 data bits. We agreed that four input buttons were enough for the HCS II project [a full alphanumeric keyboard was out!), so I used the LCD's 4-bit mode to eliminate an additional I/O port. Bits O-3 of Port 1 connect to the LCD data bus, while three Port 3 bits provide the controls.

The buttons are simple NO (Normally Open] SPST switches connected to Port 1, bits 4-7, but as you can see from Figure 1, bit 7 also drives the heartbeat LED. Although not strictly necessary, this feature allows me to work on a neat hack I've been saving for about a year: cramming two unrelated functions on one 803 1 pin. I'll describe this detail later.

ANSI CONTROLS

Unlike most "terminals" that display serial data directly on the screen, LCD-Link must coexist with other devices on the RS-485 network, which means that it must get text as part of a command and return button presses only when prompted by the HCS II SC. My approach was to graft new LCD character display routines onto the command decoder base used in previous projects in this series.

Figure 2 presents the complete LCD-Link command set, which should look familiar by now. As usual, the Q(uery) and S(how) commands will be used most heavily. The former reports button presses, while the latter is the sole means of displaying text on the LCD panel. The "A" command sets the network address; I'll discuss that in a moment.

When the SC queries the buttons, the result is returned as two hex characters. The four buttons are presented in the most significant bits, so the result for all four buttons would be FO. The button status is cleared after each Q command, so each one returns "new" button presses. The buttons do not autorepeat.

Given the LCD driver routines I've already written, transferring characters from the S command to the panel was a simple matter. However, there is no obvious way to include multiple lines in the data because network commands end with a carriage return; the first line break in the LCD data will terminate the network command!

The solution appears in Figure 3, which shows the C-style character escape sequences accepted by the LCD-Link firmware, including a line break that requires only the characters \n in the LCD data. Both are printable ASCII codes that make it easy to see what is going on. Remember that "escape sequence" in this context simply means the backslash marks an escape from the normal character definitions. It has nothing to do with the ASCII 27 "escape character" you will encounter shortly.

One complication arises when you create LCD data strings using the C sprintf() function. Because C also uses the backslash character to mark escape sequences, you must include two backslashes for each one in the final data string. For example, to clear the screen and put the words "Hello, world!" on the first two lines, use

```
sprintf(Buffer,"\\fHello,\\nworld!\\n")
```

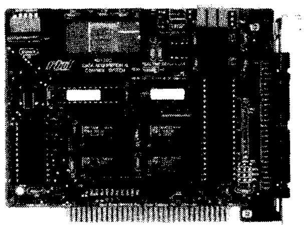
Most of the escape sequences are "just for nice" because LCD-Link will

A=string		Set network address (16 chars max, capitalized, blanks discarded)
D		Dump program status (debugging use)
E		Show and clear error flags (debugging use)
Ln		Set logging mode (bit mapped)
	L	report current mode
	LO	disable (default)
	L1	show ANSI decoding sequence
	L2	show LCD command processing
Nn		Set network/interactive mode
	N	report current mode
	NO	set interactive mode
	N1	network mode (no error messages) (default)
	N2	network mode with command echo and err msgs
	N3	same as **N1** with command acknowledgement
Q		Query buttons, only presses since last **Q** are reported
		Buttons are bit-mapped in hex byte
		Current hardware returns buttons **80, 40, 20,** 10 only
RESET		perform power-on reset must be completely spelled out!
S=string		Send string to LCD panel via ANSI decoder
		(string continues to end of line)

Figure 2-The ***LCD-Link*** firmware ***includes*** *commands* ***to query*** *the* ***button status*** *and display characters on the LCD* panel. The *other commands are* ***mostly for*** *debugging and* ***setup.***

Real Time Devices
"Accessing the Analog World"

Exciting New Products!



AD1200 - $359

- **12-bit 125 kHz A/D Conversion**
- **16 Analog Input Channels**
- **3 Timers and 16 DIO Lines**
- **Supports DMA & Pacer Clock**
- **AT Bus Version Available!**

for AMPRO CPUs

DM200 12-bit 40 kHz 8 channel analog input board; 8254 timer and 16 digital I/O lines $295

DM406 12-bit 100 kHz 16 channel analog I/O board; 2 D/A outputs; 8254 timer; 16 digital I/O lines; Supports DMA and pacer clock. $449

DM806 Hiqh current digital I/O board; 8254 timer; opto-22 compatible $195

PC/XT/AT Boards

AD2700 16 channel 12-bit 150 kHz analog input board; Supports DMA, pacer- clock and programmable gain; 8254 timer: 16 diaital I/O lines; 16-bit AT bus operation . . $525

TC48 Dual Am951 3 Timer/Counters; 24 digital I/O lines; 16-bit AT bus operation with expanded interrupts. $398

DIO24 High current digital I/O board; 24 buffered DIO lines; 8254 timer; Compatible with opto-22 equipment. $195

AD3110 Super fast 16 channel A/D board! 12-bit 200 kHz A/D rate; Burst mode operation; On board FIFO memory; Programmable gain; Timers; DIO lines; Supports DMA and pacer clock; Optional D/A outputs $665

AD3710 Low cost version of AD31 10 with on board FIFO memory and 200 kHz burst mode operation $525

DA810 8 channel 12-bit D/A; Voltage or current loop output; 8254 timer; 24 digital I/O lines $589

MR16 110 VAC mechanical relay expansion board; Computer control of 16 AC loads. **$215**

0P16 Optoisolated 16 channel digital input expansion board $225

TS16 16 channel thermocouple expansion board; Supports J & K types. ***$298***

Unparalleled ***choices*** *for real world* ***control.*** *Over* ***50*** *hardware and* ***software*** *products for single and OEM* ***users.***

FREE 80 PAGE CATALOG!

Real Time Devices, Inc.
State College, PA USA
rtd Tel.:814/234-8087
FAX:814/234-5218

#154

\cn	Control character n (\cZ = Ctrl-Z)
\e	Escape character, ASCII 27
\f	Form feed, ASCII 12
\n	New line (linefeed and carriage return)
\r	Carriage return (to current line, leftmost column)
\t	Tab to next stop: column 4, 8, 12, 16, 20
\x*nn*	Send hex char *nn* directly to LCD panel (must have two hex digits)
\\	Single backslash

Figure 3-The *"S" command string may include C-style character escape sequences to represent nonprintable control characters.*

accept the corresponding ASCII codes directly in the string. For example, you may use either \ t or ASCII 9 to produce a horizontal tab. The only essential ones are \ n and \ r, because the network interface uses those characters to delimit commands, and \ x to access oddball LCD characters.

Listing 1 shows how the firmware decodes these sequences into either LCD actions or special characters. Notice the gyrations needed for the \ x direct character output: the blank character handles the "end of line" case, then the code returns to that spot to deposit the special character. Ugh!

Using these escape sequences you can produce a fairly neat screen, but you cannot move the LCD cursor "up" or "back" to overwrite a section of the display. Because Ken intended to use the terminal to present a collection of relatively fixed items that are updated at random intervals, providing some sort of cursor control made sense.

Rather than define a whole new command set, using the standard ANSI commands makes more sense because the SC can use the same general output for a 4 x 20 LCD panel or a full 25 x 80 display on a PC. While the latter would take a bit of additional firmware, the SC wouldn't know the difference. Of course, you can't get as much on a 4 x 20 panel as a 25 x 80 display, but at least the commands look the same.

As an aside, the firmware will handle smaller LCD panels, but not the larger 8 x 40 character or IBM PC 200 x 640 pel graphic displays. If you need more area, writing code for a PC laptop makes more sense than creating custom hardware; the latter is probably more expensive than the former!

Figure 4 shows the complete set of ANSI commands accepted by the LCD firmware. Because each command starts with an ASCII 27 character, I added \e to the repertoire of the character escape sequence decoder. Thus, we have a C-style character escape sequence to create the ASCII Escape character. You can also use an actual ASCII 27 if you don't mind an unreadable character in the data string.

The ANSI command ESC [6n is an exception to the "output only" rule because it returns the current cursor location over the network. Your program must be able to extract the row and column information from the ANSI sequence you'll get back: E SC [1: 3 R means that the cursor is at row 1 and column 3. ANSI rows and columns are numbered starting from 1 rather than zero, of course.

One note of caution is in order. The LCD firmware allows up to 200 characters in any network command line, which should be enough for an 80-character display. However, overflowing this buffer is possible if you seriously overuse ANSI commands. The firmware will simply truncate the excess characters, so the display may not be quite what you expect, although no other damage will be done.

NONVOLATILE ADDRESSES

The IR-Link firmware used three Port 1 bits to select one of eight different network addresses: IRO through IR7. Allowing more than one LCD-Link terminal Would Be Nice, but the LCD and switches I/O use up all the available I/O bits. What to do?

The solution takes advantage of the Dallas Semiconductor SmartSocket battery-backed RAM socket. The DS1213C accepts either 8K or 32K byte static RAM chips and provides automatic power protection for the RAM data. When the power supply fails (or is simply turned off!), the DS1213C disables RAM write

IMAGENATION...

VIDEOFRAME
GRABBER5

- Simplicity
- Functionality
- Affordability
- Accuracy

Cortex-I $495.00

- Real-Time Capture
- Half Slot XT/AT
- 512 x 484 x 8 Bit
- RS-170/CCIR
- External Trigger

Now Cortex-STD $595.00

- Dual Video Input
- Opt. XMS Mapped
- Low Power Options
- STD-80 or 32 Bus
- External Trigger

Now Vidmux-4 $99.00

- 4 to 1 MUX
- Half Slot XT/AT

includes Software...

- C Library & Source
- Image Capture Utility
- Tiff Utilities
- "Image" Drive Ram Disk Emulation

STARTER PAK...
Cortex-Pak $1395.00

- NEC/TI-23EX Camera With Lens
- 9" Video Monitor
- Frame Grabber
- Software & Cables

OEM PRICING AVAILABLE

IMAGENATION CORP.
P.O. Box 84568
Vancouver, WA 98684

PH/FX (206) 944-9 13 1

1158

Listing 1—This code shows how the LCD driver interprets character escape sequences in the data string. Note that the C syntax for a single backslash is '\\'.

```
ANSISendString(pString)
char *pString;
{
char RepString[10];
char *pRep;
BYTE  Temp,OldRow,OldCol,NewRow,NewCol;•

  while (*pString){
    if ('\\' == *pString){                    /* escape sequence? */
      switch (tolower(*(pString+1))){
      case 'c':                               /* control chars    */
        RepString[0] = *(pString+2) & 0x1F;
        RepString[1] = 0;
        pString += 1:                         /* account for char */
        break:
      case 'e' :                              /* escape char itself */
        strcpy(RepString,"\x1B");
        break:
      case 'f' :                              /* formfeed         */
        strcpy(RepString,"\x0C");
        break:
      case 'n' :                              /* newline (LF and CR) */
        strcpy(RepString,"\x0A");
        break:
      case 'r':                               /* bare CR          */
        strcpy(RepString,"\x0D");
        break:
      case 't' :                              /* tab              */
        Temp = TABINTERVAL ((CurrentCol+1) % TABINTERVAL):
        memset(RepString,' ',Temp);
        RepString[Temp] = 0:
        break:
      case 'x' :                              /* direct hex output */
        OldRow = CurrentRow:
        OldCol = CurrentCol:
        ANSIDriver(" ");                      /* fake a single blank */
        NewRow = CurrentRow:
        NewCol = CurrentCol:
        Temp = GetHexits(pString+2,2);
        LCDSetCursor(OldRow,OldCol);    /* back to starting pnt */
        LCDSendByte(Temp,LCDWAIT|LCDDATA);/* show the byte     */
        LCDSetCursor(NewRow,NewCol);      /* to new location */
        RepString[0] = 0;
        pString += 2:                          /* account for hexits */
        break;
      case '\\':
        RepString[0] = '\\';                  /* send single slash */
        RepString[1] = 0;
        break:
      default :
        RepString[0] = *(pString+1); /* send next char anyway */
        RepString[1] = 0;
      }
      pString += 2;
      pRep = RepString;
      while (*pRep){
        ANSIDriver(*pRep):
        ++pRep:
      }
    }
    else {
      ANSIDriver(*pString);                   /* jus t copy the char */
      ++pString:
    }
  }
}
```

operations by pulling the • CE line (pin 20) high.

The SmartSocket is bright enough to avoid shutting the RAM down in the middle of the "last write" while power is failing, and it also switches to battery power before the supply drops below the RAM's lower voltage tolerance. Basically, you get a RAM that just won't quit!

Although the SmartSocket includes a battery test routine, the current firmware does not take advantage of it because the lithium battery is rated for ten years at 1 microampere drain. Just make sure that your RAM chip enters power-down mode when *CE is pulled high, and you should be all right.

The A command sets the LCD-Link network address by storing a string into RAM and computing a check value. When the power goes on again, the firmware compares the address string and the check value; if the address is valid, it's used. Otherwise [and for every power on with ordinary volatile RAM) the firmware uses the default address of "TERMO," which is stored in EPROM.

You should change the network address using an RS-232 serial terminal, because changing the address "on the fly" will probably confuse the SC! Unlike the IR-Link and PL-Link code, the LCD-Link firmware uses P1.4 to activate the interactive mode because P1.0 is used by the LCD data bus. Simply press the button while resetting the CPU and the firmware will assume it is connected to a serial terminal.

BI-DI BITS

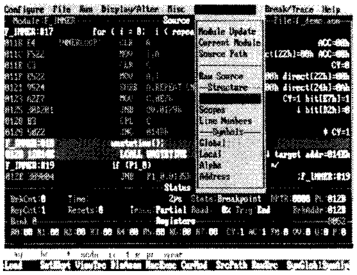
Now, for the neat hack.

Under normal circumstances an 803 1 pin serves for either input or output, but not both. For output, you simply write the data to the pin and it drives the external device either high or low. For input, you write a logic 1 to the pin's output latch and then read the input appearing at the pin. Recall that the value read by the CPU is the logical AND of the pin output latch and the external signal at the pin, so the latch must be a 1 to allow both input states.

Command	Example	Function
ESC[#A	ESC[2A	Cursor up # rows (up 2)
ESC[#B	ESC[B	Cursor down # rows (down 1)
ESC[#C	ESC[IOC	Cursor right # columns (right 10)
ESC[#D	ESC[5D	Cursor left # columns (left 5)
ESC[#;#H	ESC[H	Set cursor to row;column (to 1 ,1)
ESC[#;#f	ESC[1;2f	Set cursor to row;column (to 1,2)
ESC[#;#j	ESC[3j	Set cursor to row;column (to 3,1)
ESC[s	ESC[s	Save current cursor location (1 level)
ESC[u	ESC[u	Restore saved cursor location
ESC[2J	ESC[2J	Clear display and home cursor
ESC[K	ESC[K	Clear from cursor to end of row
ESC[6n	ESC[6n	Query current cursor location
ESC[#;#R	ESC[3;4R	Terminal's response to location query (cursor at row 3, col 4 = ESC[3;4R)
ESC[#h	ESC[7h	Set display mods (ESC[7h to wrap at end of rows)
ESC[#l	ESC[7l	Set display mode (that's an "ell") (ESC[7l to force cr/lf at end of row)
The following commands will be accepted and ignored.		
ESC[#;#m	ESC[3m	Set display attributes
ESC[#;#p	ESC[3;4p	Reassign key code

Flgun 4-The "S" command string may include ANSI cursor positioning and control stings. "ESC" represents the ASCII Escape character (ASCII27), which may be represented by the"\e" escape sequence.

8051 68HC11 COP8



ice/MASTER'"
Your Window
To Emulation
Productivity

Rental And 1 O-Day Trials Available

- iceMASTER delivers productivity: easy to learn, easy to use and fast!
- Hyperlinked On-line help guides you through the emulation process.
- iceMASTER is FAST! The 115.2K baud serial link keeps typical download times to under 3 seconds using a standard COMM port!
- iceMASTER is Versatile: iceMASTER-8051, iceMASTER-68HC11and iceMASTER-COP8 support most family derivatives.
- Flexible user interface: you can completely configure the windows for size, content, location and color.
- iceMASTER is convenient! It connects easily to your PC, requires no disassembly, nor does it take up any expansion slots. It works on any PC (DOS or OS/Z), Micro Channel or EISA. Even laptops!
- Supports source level debug (C and PL/M) and source level trace. 4Ktrace bufferwith advanced searching and filtering capabilities.
- New Products! 68HC11 A, D, E, F; 8XC552; 8XC528; 8XC515A and 8XC517A

HOT NEW PRODUCTS!

Call today for FREE DEMO DISK!
Call today to ask about FREE 8051 Macro Assembler!

(800) METAICE (800) 638-2423

MetaLink® Corporation

MetaLink Corporotion P.O. Box 1329 Chandler, Az 85244-l 329 Phone: (602)926-0797 FAX: (602) 926-1198 TELEX: 4998050MTLNK

#159

8051
Family Tools

In Circuit Emulators

The DryICE Plus is a modular emulator designed so you can get maximum flexibility from your emulator purchase. The base unit contains all the hardware necessary to support pods containing many of the most popular members of the 8051 family of embedded control microprocessors. Buy one base unit, and select one or all of the pods you need to do the job at a much reduced cost. You get the same great functionality found in our popular DryICE 8031 emulator plus real-time Execute-to-Breakpoint, Line-by-Line Assembler, and much more. And the price is (almost) unbelievable! (Yes, it works with the Mac, too!)

Base Unit (w/RS-232 IF) -- $299
Available Pods: $149 each
8031/32, 80C31/32, 80C154, 80C451, 80C535, 80C552/562, 80C652, 80C51FA, 8751/52, 87C51/52.
Call about 87C751/752 support
16K Trace Buffer option: Avail. 2nd Qtr'92
Standard 8031 DryICE -- Still only $199
Enhanced 8031 DryICE -- $269

8051 Simulation

The 8051SIM softwarepackagespeeds the developmentof 8051 family programs by allowing execution and debug without atargetsystem. The 8051 SIMulator is a screen oriented, menucommand driven program doubling as a great learning tool. $99.

Single Board Computers

8031SBC- A fast and inexpensive way to implement an embedded controller. 8031/32 processor, 8+ parallel I/O, up to 2 RS232 serial ports, +5 volt operation. The development board option allows simple debugging of 8031/51 family programs. $99ea

80C552SBC- 10 bit 8 ch. A/D, 2 PWM, 1 RS232 & 2 RS232/422/485 serial ports, sockets for 64k ROM, 64k RAM, +5 volt operation; optional RT Clock w/ battery, 2k EEPROM. Development board version available. Call for pricing!

Call for your custom product needs.
Free Quote - Quick Response

VISA MasterCard AMERICAN EXPRESS Card

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

(619) 566-1892

#160

A few special cases, like the LCD panel data bus, call for bidirectional I/O. The firmware must ensure that the pin latch is set to 1 whenever data is arriving from the external device, but doing so is straightforward because the firmware controls what the LCD is doing and data never arrives unexpectedly. If the external device can source a large amount of current, you can damage the 803 1 hardware by trying to pull a pin down to ground with a 0 output, so you must be careful to ensure valid controls.

However, using the heartbeat LED is a completely different situation. The pin output latch switches between 0 and 1 every second, but the switch may be pressed at any time. The firmware must scan it many times a second to capture the switch value, but half the time the 0 output overrides the input.

Obviously, because of the way the hardware works, you must set the latch to 1 before reading the input, then restore the original value afterward. The catch is all of the ordinary "read the port" instructions (e.g., MOV A , P 1) return the latch state ANDed with the input signal. When the output is 0, the CPU cannot tell whether the input is 0 or 1.

Unlike many CPUs, the 803 1 includes a wide variety of bit manipulation instructions. These instructions can be applied equally well to both internal CPU bits (such as the ALU

Listing 2-The state *of an 8031 I/O pin* is he *logical* AND *of the output latch and the* input *signal:* if the *output is 0, the pin will always be read as 0. This code section shows how to record the latch state, set it high, read the actual* input signal, and *then* restore *the latch.* This *trick cannot be used for pulse outputs because it causes a low* g/itch each time the *pin is read.*

```
        JB    FlagPortBusy.swdone ; skip if port is in use

        MOV   B.#$FF              set up output latch memory

        JBC   P1.7 sw?7           branch if latch set. clr latch
        CLR   B.7                 latch was off. record in B
sw?7    JBC   P1.6. sw?6
        CLR   B.6
sw?6    JBC   PI.5 sw?5
        CLR   B.5
sw?5    JBC   P1.4.sw?4
        CLR   8.4
sw?4    JBC   P1.3.sw?3
        CLR   8.3
sw?3    JBC   P1.2.sw?2
        CLR   B.2
sw?2    JBC   P1.1.sw?1
        CLR   B.l
sw?1    JBC   P1.0.sw?0
        CLR   B.0
sw?0    EQU   *

        MOV   P1.B                ; restore latches
        ORL   P1 .#INPUTMASKASM   ; set input bits high for read
        MOV   A.P1                ; fetch inputs

        MOV   P1.B                ; restore latches again

        CPL   A                   ; convert to pos. logic (1=ON)
        ANL   A.#INPUTMASKASM     ; isolate input bits
        MOV   B.A                 ;save for later update
        XRL   A.KeysLast          ; compare with previous sample
        MOV   KeysLast.B          ; and update previous sample
        JZ    swdone              ; if zero. nothing changed

        ANL   A.B                 ;1 bits = new ON buttons
        ORL   KeysHit.A           ; record the fact...

swdone  EOU   *
```

Carry flag) and external inputs (such as Port 1 bits). Becoming familiar with these instructions will help you write better firmware-and in this case, one instruction will help you achieve the seemingly impossible.

The J BC instruction tests a bit, clears it to 0, then branches if the bit was originally 1. Normally you would J BC on a status flag set by one routine, typically an ISR, and used by another, typically mainline code. Because the test-and-clear takes place within single instruction, an interrupt handler cannot "get in between" to create an invalid condition.

However, when applied to I/O pins, J BC tests only the output latch, not the "latch AND input data" used by the other instructions. The J BC will branch if the output latch was 1 and fall through if the bit was 0. In either case, the output latch will go to 0. See how this instruction comes in handy!

Listing 2 is extracted from LCD-Link's Timer 0 interrupt handler. The timer ticks every 5 ms, so the switches are sampled 200 times each second. This amount is admittedly excessive, but the CPU generally isn't doing much else and it was easy to keep the timer tick at about the same rate for all these projects.

The HCS II SC must poll the LCD-Link to determine which switches were pressed, so the firmware uses two global variables to record both the previous state of the port and each new key press as it occurs. The firmware clears the key states, so each Q command gets new presses.

Because J BC clears the bit after testing it, the output device must be able to withstand a short glitch to 0 whenever the bit was originally 1. In the case of the heartbeat LED, this action causes only a dim glow when the LED should be off (remember that a LOW output is ON), but devices like relays may object to the mistreatment.

A further complication arises when the port bits are shared between several unrelated functions. For example, consider what would happen if the LCD driver had set up a command in the low order bits and activated the control lines to send it to the LCD panel when a timer interrupt glitched the port bits. The LCD panel objects to data glitches while the control lines are active [and rightfully so!), so we must ensure that they occur only when the LCD panel control lines are inactive.

The solution is a semaphore that prevents the timer code from sampling the input port when the LCD is busy. The first statement in Listing 2 tests Fl agPortBusy, which the LCD code sets while it is using the port. This feature does not interfere with reading the switches because the LCD is generally idle and LCD hardware operations are quite brief. In any event, with 200 samples per second, the switches are read often enough!

In summary, there are a few special cases where you can use an 8031 I/O bit for unrelated input and output functions, but you must be careful about side effects. That you cannot get this level of bit-banging in C should also be apparent; there is still room for assembly language!

RELEASE NOTES

The BBS files include LCD-Link's executable EPROM hex file and the source code sections shown in the listings. The complete LCD-Link source code may be licensed from Circuit Cellar Inc. (not INK). ❑

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

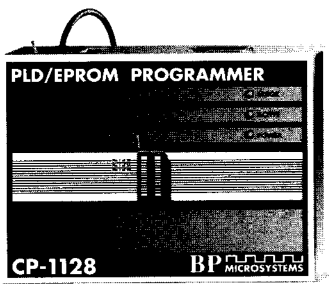
SOURCE

HCS II components are available in both kit form and assembled and tested. Contact:

Circuit Cellar Kits
4 Park St.
Vernon, CT 06066
(203) 8752751
Fax: (203) 872-2204

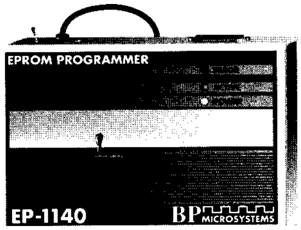
I R S

416 Very Useful
417 Moderately Useful
418 Not Useful



CP-1 128
Combination Programmer
$1295.00

✔ Supports AMD's MACH110/210/120/130/230 EPLDs, Altera's 900- , 1800-series and MAX EPLDs, Cypress' CY7C361, Lattice's isp I511032 & pLSI1032, National Semiconductor's -5D & -7D devices and MAPL devices.
✔ Qualified and recommended by AMD, Lattice, National Semiconductor, Signeticsand others.
✔ Utilizes only manufacturer approved programming algorithms.
/Supports upto 28-pin E/EPROMs and bjpolar PROMs including the microwire I²C devices.
/Supports Dallas Semiconductor NVRAMs and TI DSP320, Microchip PIC microcontrollers.
✔ Lifetime FREE software updates available via BBS and US Mail.
✔ Call for a DEMO disk and literature pack.
✔ Made in the USA



EP-1140
E/EPROM & µcontroller
Programmer
$695.00

✔ The EP-I 140 supports NEC's 27C8001, 8-Mbit EPROM, all 27C240 4-Mbit 16 bit EPROMs, FLASH EPROMs, NVRAMs and all microwire EPROMs.
✔ All Intel, AMD, and Signetics 40-pin µcontrollers supported directly.
✔ Qualified and recommended by Intel. Signetics, National Semiconductor, and others
/Lifetime FREE software updates available via BBS and US Mail.
✔ Risk-free thirtyday moneybock guarantee.
✔ Made in the USA

With lifetime FREE software updates, you can't go wrong and BP Microsystems offers a thirtyday money-back guarantee to ensure product satisfaction. Remember, BP Microsystems is...

The Engineer's **Programmer**™

BP MICROSYSTEMS
1068 1 Haddington • Houston, TX 77043-3239
(713) 461-9430 • FAX (713) 461-7413

© 1991 BP Microsvstems Inc

#161

Computers on the Brain (revisited)

FROM THE BENCH

Jeff Bachiochi

Circuit simulation is becoming a more affordable design tool every day. Get inside Jeff's head as he takes a new look at a popular Ciarcia project.

i’m continually amazed at the staying power of certain projects, particularly the “Hemispheric Activation Level detector” introduced by Steve in the June 1988 issue of BYTE. Here it is, four years later, and HAL’s popularity has not decreased one bit. Sometimes a project you were involved with comes back from the dead to haunt you. Sometimes, as in HAL’s case, it haunts you continuously from the beginning.

Like many of Steve’s projects, HAL is a bit unusual. With HAL, Joe Average now had the ability to monitor brain activity in the 4- to 24-Hz region (theta, beta, and alpha waves). HAL’s input comes from electrodes placed at various positions on a subject’s head (most commonly on the front and rear of each hemisphere). Microvolt input signals are amplified and then sampled by an ADC. Finally, the samples are transmitted in a serial bit stream to a host computer. HAL operates on batteries and the serial link is optoisolated, preventing any possibility of electrocution. The host, any computer that will accept RS-232, is responsible for storing the transmissions and analyzing and displaying the results, all in real time.

DATA FLOOD

A tip of the hat to the software gurus Robert Schenck and Dave Shultz. The timings involved within HAL and the host must be matched precisely to allow the FFT software to dissect HAL’s sampled composite waveform data into individual components of frequency and amplitude for each channel. An example of host software was written for the IBM PC that displays two of the four channels HAL actually transmits. This sample code was meant as a starting point for experimenters, each of whom has their own ideas of what should be done with HAL’s continuous deluge of 4800-bps data. Developing code for HAL requires expert programming skills. Fortunately, you can study the example software and shorten the learning curve by taking advantage of the source code’s availability.

HAUNTING HARDWARE

Having the source does eliminate many questions and minimizes software support. It does not reduce the questions about the hardware and what can be done to change the specifications. Let me provide a look at HAL’s specifications and at the design of the analog front end. I believe this information will help to answer many questions on the potential flexibility of HAL.

Analog front-end specifications are as follows:

*Input DC current < 50 nA
*Equivalent input noise level of <0.5 μV
*Flat bandwidth between 4 and 20 Hzoffl db
•18 db/octave roll-off (–50/60 db at 60 Hz)
*Minimum input detection of 5 μV

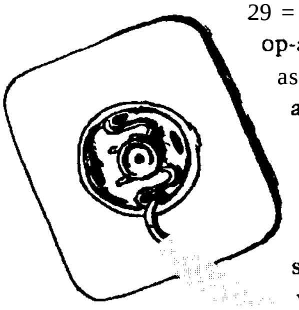
From these specs you can determine that HAL must pick up and amplify microvolt signals between 4 and 20 Hz while rejecting noise and other signals outside of the passband. Refer to Figure 1 for the analog front end of HAL. Notice I’ve broken down the front end into stages, so you can investigate them by reverse engineering at each stage.

All op-amps used are Texas Instruments’ TL084s, which have JFET inputs, extremely low picoampere input bias currents, and a low cost. The quad packaging, keeps the real estate down to a minimum.

The first three op-amps in each channel’s input circuitry are used as an instrumentation preamplifier. Two noninverting op-amps create a high input impedance front end with their negative inputs connected together rather than to ground as in a normal noninverting amplifier. The gain of each amplifier is

$$1 + \frac{R3a}{Rla} = 1 + \frac{470k}{16k} = 29$$

The total gain for this stage of the instrumentation preamplifier is the sum of each front end amplifier, 29 + 29 = 58. The third op-amp is used as a differential amplifier. Signals common to both electrodes, such as noise, will be amplified equally by amplifiers #1 and #2 and canceled by the differential amplifier #3, regardless of the gain set and as long as the inputs are equal. Signals produced by the brain will be unequal unless the electrodes are equidistant from the source, so the outputs from amplifier #1 and #2 will be different. Differential amplifier #3 will amplify the difference by a gain of



$$\frac{R6a}{R4a} = \frac{510k}{5.1k} = 100$$

The total gain for the instrumentation preamplifier stage is 58 x 100 = 5800 at the f_c (center frequency)

I use the term f_c because Cla and C2a remove any DC offsets from amplifiers #1 and #2, acting as a high-pass filter that has an f_{co} (cutoff frequency) of

$$\frac{1}{2x\ (R5a)(C2a)} = \frac{1}{2(3.14)(5.1k)(3.3\mu F)}$$

$$= 9.5 \text{ Hz}$$

with a -3-db attenuation per octave. Feedback capacitor C3a acts as a low-pass filter with the differential amplifier stage that has an f_{co} of

$$\frac{1}{2\pi\ (R6a)(C3a)} = \frac{1}{2(3.14)(510k)(0.01\mu F)}$$

$$= 31.2 \text{ Hz}$$

with an attenuation of -3 db per octave.

Between the instrumentation amplifier and the first active filter section is a passive RC high-pass filter. This filter eliminates any DC offset from the previous amplifier section and has a calculated f_{co} of

$$\frac{1}{2\pi\ (R8a)(C4a)} = \frac{1}{2(3.14)(10k)(5.6\mu F)}$$

$$= 2.8 \text{ Hz}$$

with a -3-db per octave attenuation.

The first filter stage is a 3-pole Butterworth low-pass active filter. The f_{co} is calculated as

$$\frac{1.392}{2x\ (R9a)(C5a)} = \frac{1.392}{2(3.14)(100k)(0.1\mu F)}$$

$$-22 \text{ Hz}$$

with an attenuation of -18 db per octave.

Between the two filter sections is a passive RC high-pass filter. Again, this feature eliminates DC offset from the first filter section and has a calculated f_{co} of

$$\frac{1}{2x\ (R12a)(C8a)} = \frac{1}{2(3.14)(4.7k)(10\mu F)}$$

$$= 3.4 \text{ Hz}$$

with an attenuation of -3 db per octave.

The second filter section is identical to the first and as such has the same characteristics, an f_{co} of 22 Hz with an attenuation of -18 db per octave.

Finally, a high-pass filter with an f_{co} of 3.4 Hz removes the DC offset from the second filter section and feeds the composite AC signal into the final amplifier for a gain of 2. This amplifier's positive input is biased at 0.5 Vcc. The resultant amplifier's output is at a DC level of 0.5 Vcc with the composite AC signal swinging about that DC level.

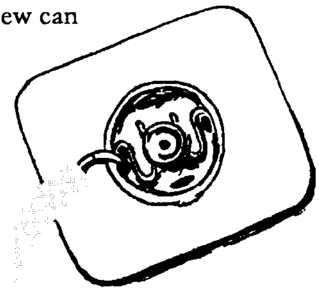
A 2.5-volt reference diode with equal tail resistors between Vcc and ground provide the ADC with a positive reference of 1.25 volts above 0.5 Vcc and a negative reference of 1.25 volts below 0.5 Vcc. With only the DC bias of 0.5 Vcc and no AC input to the ADC, the A/D count will be approximately 128, or half the full scale. When an AC signal is present, the count will rise above and fall below 128 in proportion to its amplitude. The reference voltage (2.5 V) divided by the resolution for the ADC equals about 10 mV per count, meaning if the AC signal is 20 mV peak-to-peak, then the count will be 128,129, 128, 127, 128 for one AC cycle. Because the overall gain of the front end is 58 × 100 × 2 or 11600, if you divide 20 mV by 11600, you get the minimum size the input signal must be for the ADC to recognize it as one count: 1.7 µV.

If I graph each of these calculated sections, I end up with the overall response shown in Figure 2 (theoretical).

SPICE IT UP

Whether your poison be solderless breadboards, wire-wrapping, or the sweet smell of solder, there is nothing like "hands-on" hardware for trying out a new circuit idea. Although solderless breadboards are the quickest to use, they are not suitable for many kinds of applications nor do they give the permanence of the latter two choices. But what other alternatives are there?

Circuit simulation today is where PC-board layout packages were ten years ago: few can afford it. However, like all good products, competition is driving down simulation's prices while raising the level of performance. Simulation of digital and analog circuits is now becoming affordable, and allows the computer to simulate not only circuitry but equipment such as power supplies, sweep generators, meters, and scopes-the kind of equipment you ordinarily might not be able to afford.



Many "Spice-like" programs require only a list of circuit components that contain type, value, and connection information. Some schematic and layout packages have integral simulators built in, and the input information is passed from the

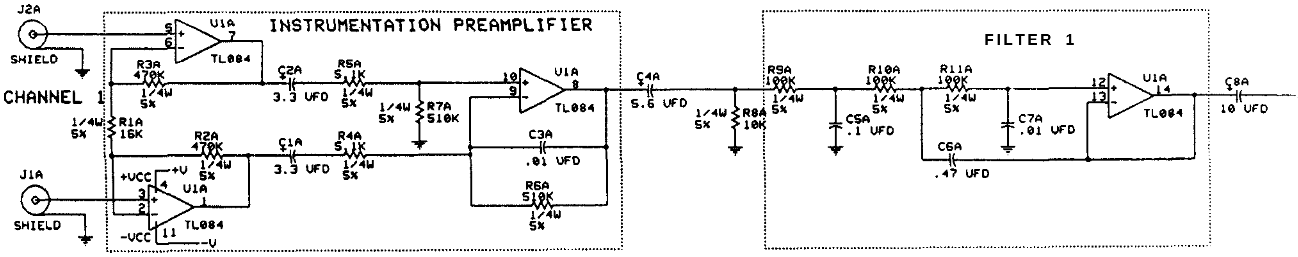


Figure 1—*HAL's analog* front end *consists of numerous amplifiers and filters. The other three channels are identical.*

schematic module to the simulation module. If you don't have one of these, you can still do simulations. Let me introduce one of these packages to you. B^{2}Spice [for Windows) from Beige Bag Software. I am fighting tooth and nail to stay away from Windows, but this software is so easy to use that it makes putting up with Windows' many aggravations is almost worthwhile.

I entered the schematic of HAL's front end into the simulator by simply choosing a component, placing it, entering its value, and making the appropriate connections to other components. I used the minimum input necessary (2 μV), which should give an output of about 20 mV. Next, I will show what happens to the circuit with an AC input from 0.1 Hz to 1000 Hz.

I placed the appropriate "signal generators" and "voltmeters" into the schematic, then started the simulation by entering the starting and ending frequencies as well as the number steps of simulation per decade to get a table, a graph, *or* both, of calculated points. Because gain is not a factor at these low frequencies, I did the circuit simulation using the ideal op-amp from the component list and not a TL084-type model.

The ease of twiddling values and getting back simulation results almost immediately is truly addictive (caution: for this reason, simulation should only be used in small doses). I found myself playing the "What if?" game with component values while hours slipped away!

Affordable 8031 Development

Single Board Computers, Assemblers, Compilers, Simulators, and EPROM Emulators

Control-R Series, Single Board Computers

Two models of Control-R series computers make pmtotyping, one of a kind products, or small production runs easy and economical. Both feature RS232 compatible serial ports, single 5 volt supply operation, and direct access to Ports 1 and 3 of the 8031. Additional features are as follows:

Control-R Model 1 $49.95

Fully populated board with I/O header for Ports 1 and 3, serial port, and 8K EPROM socket. 3.0" x 4.0"

Control-R Model 2 $79.95

Same features as the Control-R 1 plus 8K of SRAM and expansion bus with data, address, RST, INT1, WR, RD, PSEN, ALE and T1. 3.5" x 4.5"

Software and Hardware Development Tools

Control-C 8031 Cross-Compiler $200.00

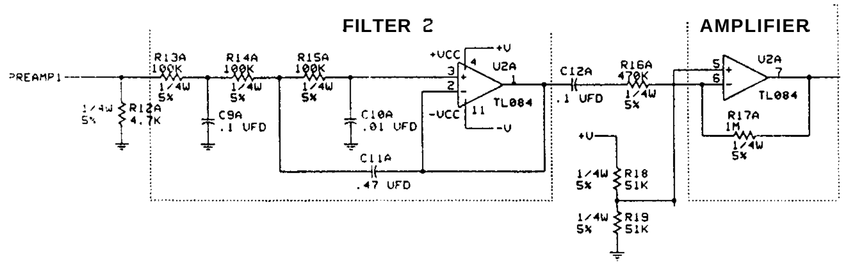
The Control-C 8031 cross-compiler is a full featured K&R style C development system available at an affordable price. Optimized for embedded system use, it will produce ROMable code for any 8051 based system including designs using only the 128 bytes of internal RAM. Package includes compiler, pre-pmcessor, assembler, simulator, printed documentation and complete library source code. Requires IBM PC or compatible. 5.25", 360K disk.

PROMulator 256 $189.95

An EPROM emulator lets you avoid "Bum and Test" development cycles. In circuit emulation of 2K-32K 27xx series EPROMs. ABS Plastic case. Assembled or compiled code is downloaded directly to the target hardware.

Cottage Resources Corporation
Suite 151, 10271 South 1300 East
Sandy, Utah 84094
VISA/MC, COD. Call to Order: (801) 268 - 2875

#162



Confidence is built when the simulator results (Figure 3) are compared to the theoretical calculations done previously. So much for theory and simulation; what about reality? What about the the real thing?

FROM THE BENCH

Although HAL is meant for amplifying brain waves, the thought of sticking electrodes on my scalp while trying to measure stage gains seemed self-defeating. I decided to use artificial input: a signal generator.

The first problem I ran into was two-fold: there was too much signal strength, and the generators had a low output impedance (50 ohms). A buffer op-amp took care of both. I placed a 15M/15k resistor divider on the buffer's output to approximate the electrode impedance and get the signal down into the microvolt range. At this amplitude you can't see the signal for the trees, ah, noise. It's really hidden deep in the forest!

With an oscilloscope, I made output measurements of each stage, listing the amplitudes at various frequencies. After all the measurements were made, I regraphed by decibels versus frequency, similar to the previous ones, so they could be compared easily. I checked the slopes and found the upper slope to be about right, 39 db per octave. However, the lower slope seemed a bit steep, which bothered me because there shouldn't be any more than a 24-db roll-off here. In rechecking the setup, I noticed the signal generator was not linear below 5 Hz; it rolled off at about **12** db per octave. After compensating for the input error, the plot better resembled what I had originally expected. If you compare Figure 4 with the previous two, you will see that all are indeed similar.

THE BOTTOM LINE

Initial specifications for input DC current <50 nA are met by using the TL084 op-amp. The TL084's FET inputs are a high 10^{12} ohms with input offset and bias currents of 5 pA and 30 pA, respectively. Bandpass specifica-

THE $99.95
EMBEDDED EDUCATION

THE PRIMER MICROPROCESSOR
TRAINING SYSTEM

TEACHES:
* INTEL 8085 PROGRAMMING
. DIGITAL & ANALOG INTERFACING
* PROGRAMMING INTEL PERIPHERALS
* MICROCOMPUTER DESIGN & ASSEMBLY

FEATURES:
* MONITOR O.S. SOFTWARE IN EPROM
. OVER 100 PAGE SELF INSTRUCTION MANUAL
. 6 DIGIT, 7 SEGMENT, LED DISPLAY
* 20 KEY KEYPAD
* DIGITAL INPUT PORT WITH DIPSWITCH
*DIGITAL OUTPUT PORT WITH LEDs
. ANALOG TO DIGITAL CONVERTER
* DIGITAL TO ANALOG CONVERTER
'TIMER/COUNTER WITH SPEAKER OUTPUT

OPTIONS:
* BASIC OR FORTH LANGUAGES IN EPROM
* RS232 SERIAL PORT CONNECTS TO PC
* BATTERY BACKED CLOCK AND RAM
. 9 VOLT 500 MA. POWER SUPPLY

EMAC OFFERS A COMPLETE LINE OF MICROPROCESSOR TRAINING SYSTEMS STARTING AT $99.95 QUANTITY 10 FOR THE PRIMER KIT.

EMAC, inc.

618-529-4525 FAX: 618-457-0110
P.O.BOX 2042 CARBONDALE, IL 62902

#163

Cross-Assemblers from $50.00
Simulators from $100.00
Cross-Disassemblers from $100.00
Developer Packages
from $200.00 (a $50.00 Savings)

Make Programming Easy
Our Macro Cross-assembten are easy to use. With powerful conditional assembly and unlimited include files.

Get It Debugged - FAST
Don't wait until the hardware is finished. Debug your software with our Simulators.

Recover Lost Source!
Our line of disassemblers can help you re-create the original assembly language source.

Thousands Of Satisfied Customers Worldwide
PseudoCorp has been providing quality solutions for microprocessor problems since 1985.

Processors

Intel 8048	RCA 1802.05	Intel 8051	Motorola 6805
Motorola 6800	Motorola 6801	Motorola 68HC11	WDC 65C02
Hitachi 6301	Motorola 6809	MOS Tech 6502	NSC 800
Rockwell 65C02	Intel 8080, 85	Zilog Z80	
Hitachi HD64180	Mot. 68k, 8,10	Intel 8096, 196kc	

New
Zilog Z8
Zilog Super a
• All products require an IBM PC or compatible.

For Information Or To Order Call:
PseudoCorp
716 Thimble Shoals Blvd., Suite E
Newport News, VA 23606
(804) 873-1947
FAX: (804) 873-2154

141

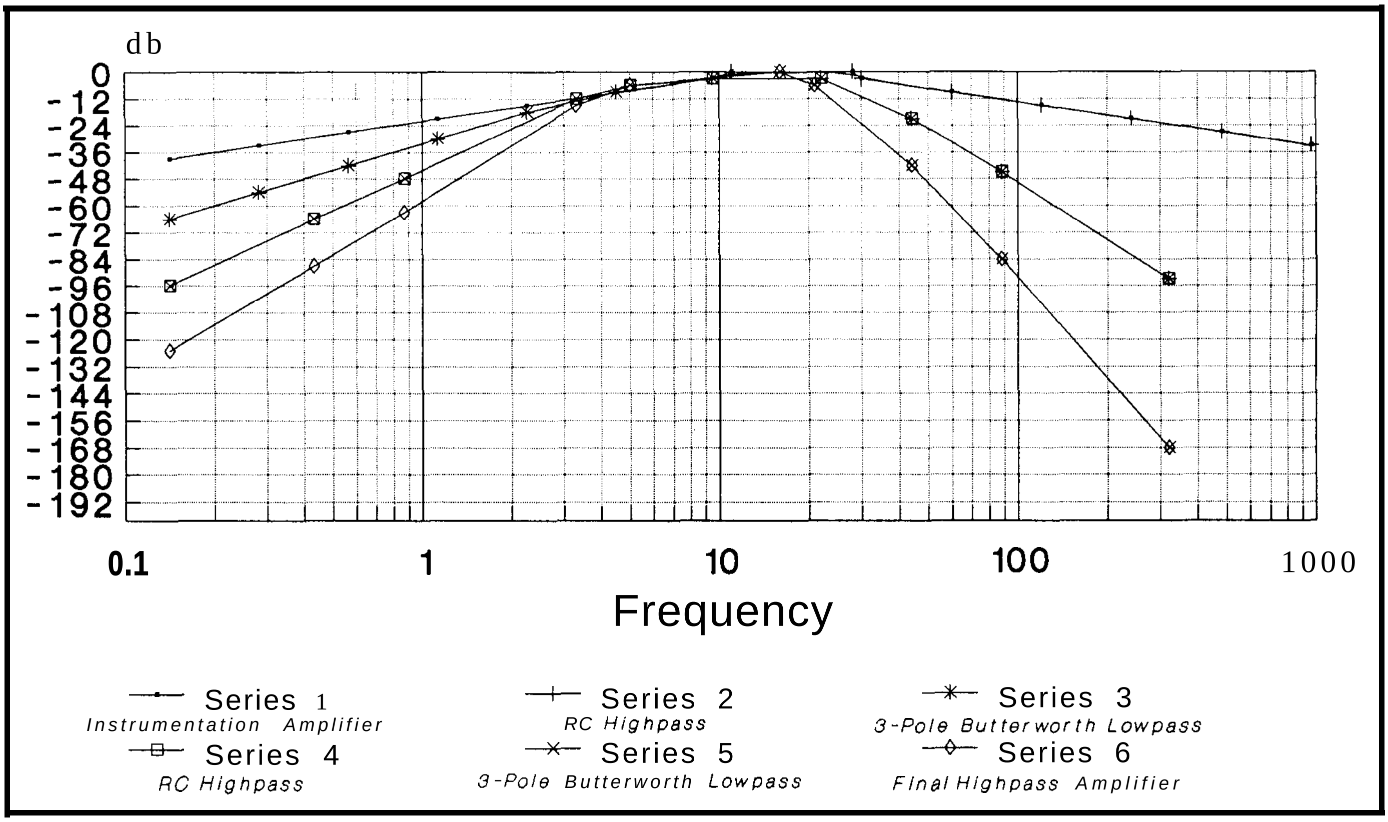


Figure 2—*Equations and a calculator* give *the theoretical frequency response of HAL's front end.*

tions are between 4 and 20 Hz. As you can see from the expanded view in Figure 5 (passband), the actual (present) -3-db passband is about 12 Hz to 20 Hz. However, this range does not quite meet with passband specifications. The lower f_{co} could be improved by increasing C2 to 15 µF (f_{co} from 9.5 Hz to 2 Hz). The upper f_{co} could be improved by reducing R9-11 and R13-15 to 75k (f_{co} from 22 Hz to 30 Hz). These adjustments would reduce the

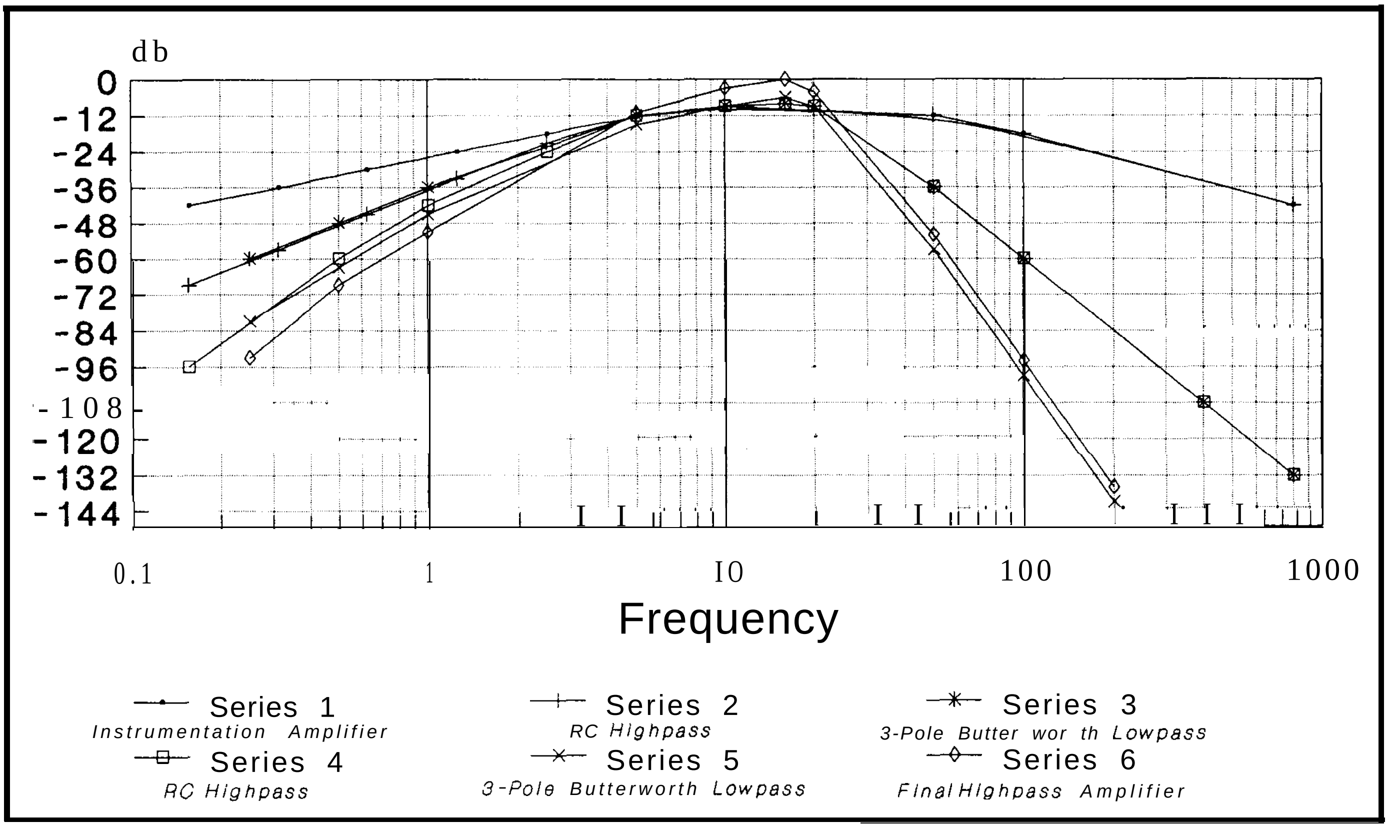


Figure 3—*Running HAL's front end through an analog* circuit simulator *gives results similar to tie theoretical calculations.*

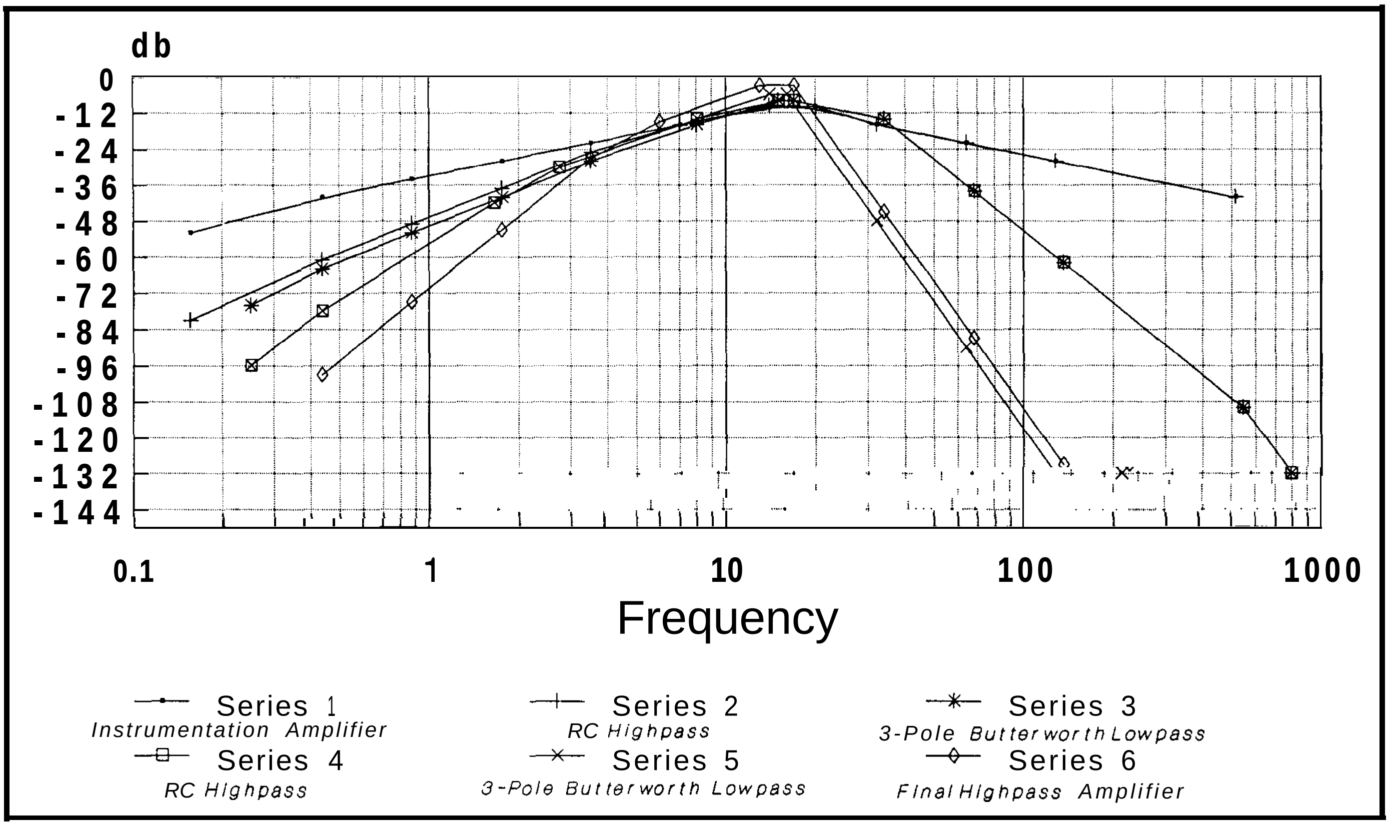


Figure 4—*Measuring* key *points on the* circuit *in action confirms both the theoretical and* simulated *results.*

60-Hz attenuation by about 9 db. The actual attenuation of 60 Hz is about 63 db. Even with the 9-db reduction, 60-Hz attenuation falls within specifications. By modifying six resistors and two capacitors, you improve the passband dramatically.

As demonstrated here, passband specifications can be altered by adjusting the appropriate sections of HAL's front end. Adjusting components in stages 1, 2, 4, and 6 alters the lower f_c. Adjusting components in stages 1, 3, and 5 alters the upper f_c. In order to remain within the physical limitations of HAL's layout, some sense of part size must be realized. Just because you can calculate a component value does not necessarily mean it will fit in the existing space provided. In addition, larger capacitors may not be available with the same lead spacing, body diameter, or tolerance.

Resistors are easily available in 5% or even 1% tolerances, but capacitors are most commonly found with tolerances as high as 20%. Because you and I would like all four of HAL's channels to be accurate not only on their own, but with respect to each other, the actual capacitor values must be held to at least a 5% tolerance. This limit can be met by purchasing capacitors already closely matched (5% or less tolerance) or by selecting capacitors yourself, using a capacitance meter.

CONCLUSION

I may be a bit naive to think this article will cut down on the number of HAL support calls that I field. It probably raises more questions than it answers. But isn't that what life's about-the search for answers? ❑

REFERENCES

Steve Ciarcia, "Ciarcia's Circuit Cellar-Computers on the Brain, Part 1," BYTE, June 1988.

Steve Ciarcia, "Ciarcia's Circuit Cellar-Computers on the Brain, Part *2," BYTE,* July 1988.

Willis J. Tompkins &John G. Webster (editors), *Interfacing Sensors to the IBM PC,* Prentice Hall, 1988.

Thanks to Chris White for his assistance with the graph presentation using Harvard Graphics.

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

SOURCES

Beige Bag Software
715 Barclay Ct.
Ann Arbor, MI 48 105
(3 13) 663-4309

HAL kits are available from

Circuit Cellar Kits
4 Park St.
Vernon, CT 06066
(203) 875-2751
Fax (203) 872-2204

I R S

419 Very Useful
420 Moderately Useful
421 Not Useful

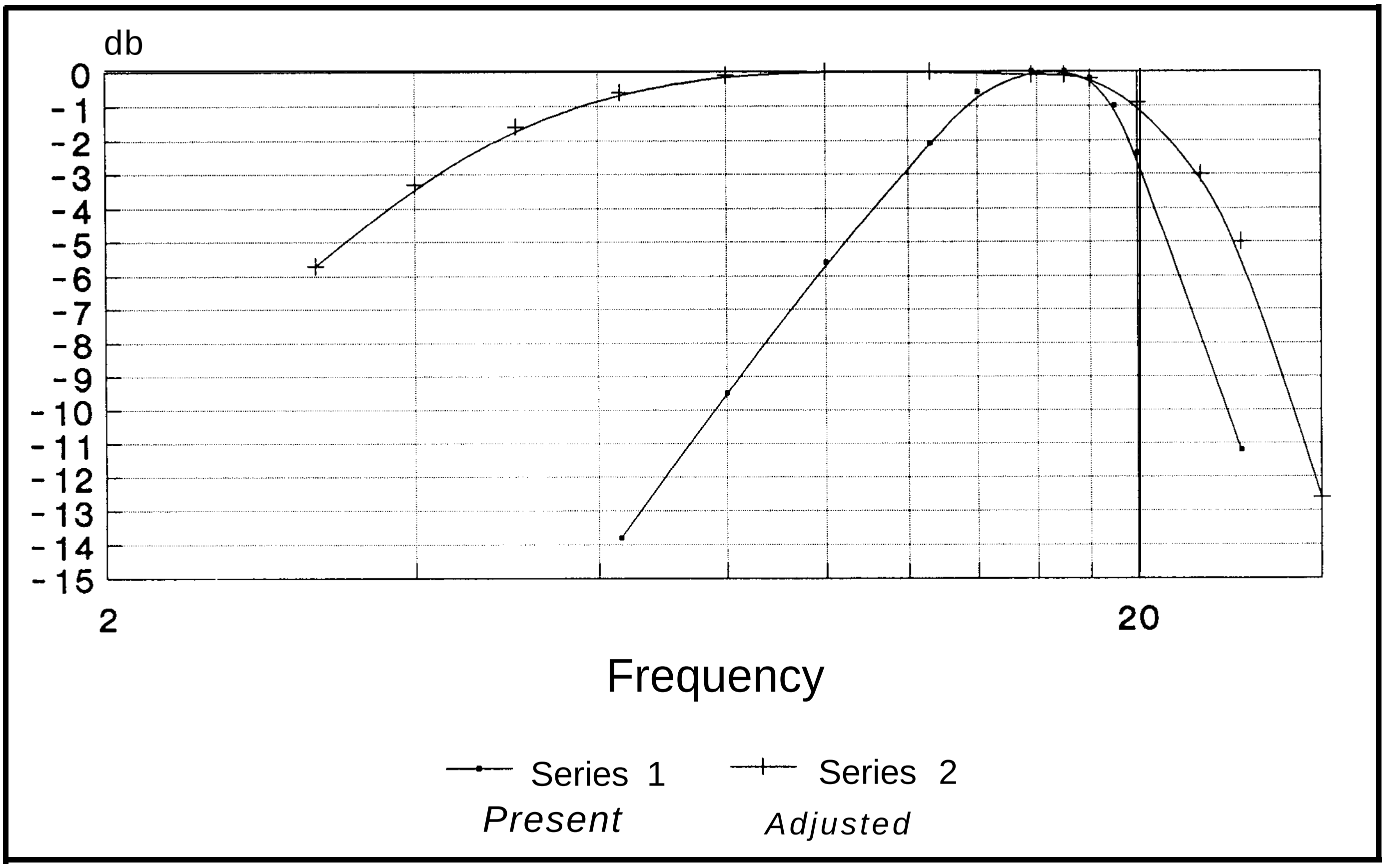
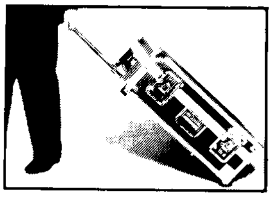


Figure 5—*By changing the values of six resistors and two capacitors, the passband of he original circuit* (series 1) can be *greatly* expanded (series 2).

Take It Easy.

Take it easy on your cargo with a custom Cabbage Case built to the exact dimensions of your equipment.

Take it easyonyour back with our extension handle and tilt wheels options. Take it easy on your wallet. Let Cabbage Cases show you how easy it is to save money on quality, custom-built road cases that make shipping and traveling with your valuable cargo safer and easier. **Prices quoted over the phone. Call 800-888-2495 today.**



CABBAGE CASES, Inc.

1166-C Steelwood Rd.
Columbus, OH 43212
800/888-2495
614/486-2495
FAX/486-2788

165

REMOTE POWER CARD!

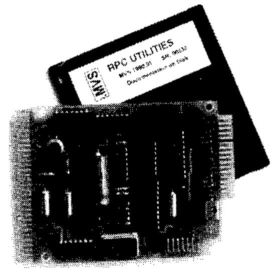
3 VERSIONS:

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

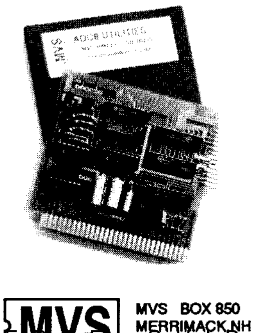
PHONE
TURN ON PC WITH PHONE, SHARE VOICE / MODEM LINE, CONTROL AC APPLIANCES

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

**95$** 27$ OEM



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



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

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

MVS
MVS BOX 850
MERRIMACK NH
(508) 792 9507

5 YEAR LIMITED WARRANTY
FREE SHIPPING IN USA

166

Multimedia Madness

SILICON UPDATE

Couch Potato Computing

RISC, CISC, PIC, fuzzy logic—Tom has never been one to shy away from the latest buzzword. This time, he's tackling something we've all been hearing a great deal about: multimedia.

Tom Cantrell

The future is clear. The difference between audio, video (also known as A/V), and computing has blurred and will eventually disappear. That's right, someday "Superboxes" will incorporate all the functionality of today's curious and incompatible mix of TVs, stereos, computers, video games, and so forth. Ultimately, the barrier to the Superbox may be misguided marketing strategies rather than technology. Isn't selling lots of assorted proprietary boxes, media, and cables, all with built-in obsolescence, better than a single, multifunctional product?

Seeing who will ultimately drive the market will be interesting: the current A/V suppliers (e.g., Sony), computer suppliers (e.g., Apple), or new hybrids (e.g., Sony + Apple].

While the Superbox is clearly located at our point of destination, getting there from here is another story. In fact, right now we're at that ugly phase all infant markets go through, characterized by multiple suppliers using multiple technologies with multiple standards pursuing multiple customers who have multiple applications. One byproduct of the confused market state is an explosion of acronyms, each claimed by its proponent to be the multimedia Holy Grail, such as CD-I, CD-DA, CD-XA, CDTV, DVI, PVI, AVK, DCT, MPEG, JPEG, IMA, MPC, PCM, MIDI, PAL, SECAM, NTSC, RGB, VI-IS, VISCA, MTS, UHF, VHF, FCC, AVSF, PCS, CCIR601, H.261, and Px64.

Oh, well. Nobody said the trip would be easy. Maybe the best way to define multimedia is to follow Associate Justice Potter Stewart's lead when he said of obscenity: "I shall not today attempt further to define the kinds of material...but I know it when I see it."

Anyway, Superbox-capable silicon is right around the comer, and now is the time to start checking it out.

PEG IN A BLACK HOLE

Because I have a day job, I won't be able to cover all the above buzzwords here. Instead, I'll try to hit a few of the high notes and give you my opinion of what's hot and what's not.

By far, the fundamental shift of information from the analog to the digital realm has revolutionized multimedia at its heart. As with any revolution, there are winners and losers; for example, those that have invested in "analog" HDTV (high-definition television] schemes will have to go back to the drawing board.

Digital schemes also face a big challenge, namely "so many bits, so little bandwidth." Consider that broadcast-quality video [say 640 H x 480 V x 24 bits per pixel x 30 frames per second) requires an astounding 36.8 megabytes per second! Obviously, the storage speed and capacity requirements are somewhat problematic, amounting to a bandwidth "black hole." Don't expect to fit Hollywood's latest overblown epic on your dinky hard disk anytime soon.

So, we're clearly not going to be able to force the issue, despite the inevitable improvement in storage price and performance. Instead, the situation calls for some finesse in the form of A/V "compression."

To achieve this end, various "PEG" standards are being developed including JPEG (Joint Photography Experts Group], MPEG (Motion Picture Experts Group], and MPEG 2 under the auspices of ANSI/ISO (American National Standards Institute/International Standards Organization).

These models are differentiated by the type of material being compressed and the tradeoff in demands placed on the compression scheme and the storage devices. JPEG, which has achieved "draft" standard status, is designed for single or still images,

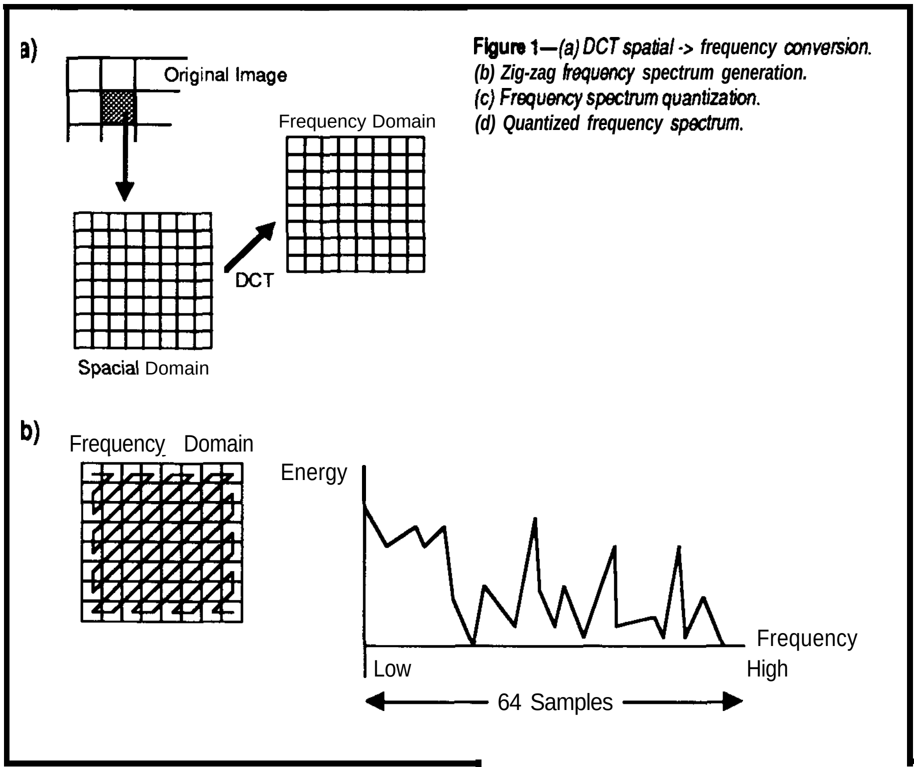
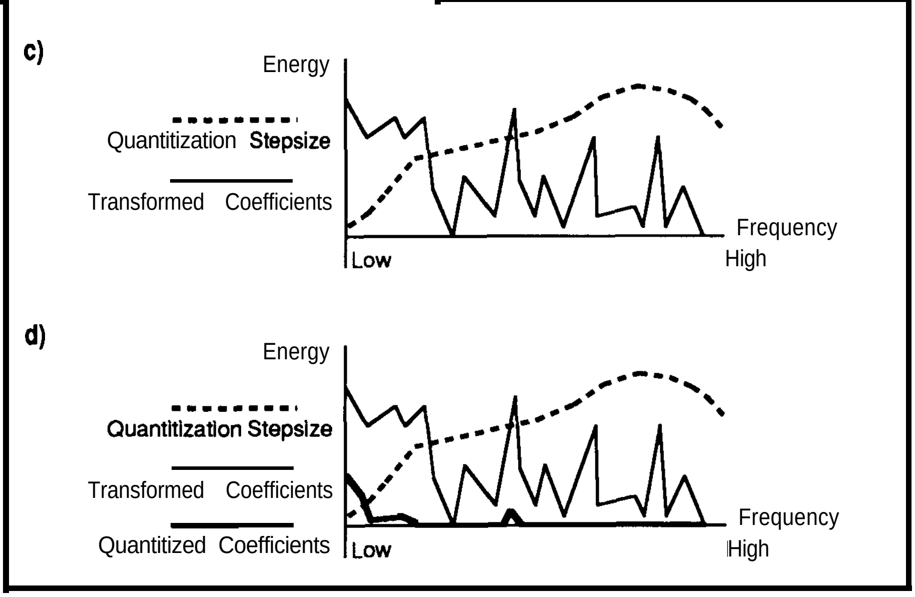


Figure 1—*(a) DCT spatial -> frequency conversion.*
(b) Zig-zag frequency spectrum generation.
(c) Frequency spectrum quantization.
(d) Quantized frequency spectrum.

while MPEG targets motion pictures as its name implies. MPEG has achieved proposal status and targets minimal bandwidth, namely 1.5 megabits per second or so; within the capability of CD drives. While MPEG 2, which is just now being debated (40 initial proposals!), promises high-quality video at the expense of more, but (hopefully) still feasible, bandwidth that will be within the realm of PCs, hard disks, LANs, and so forth.

Meanwhile, CCITT also has its hat in the ring with the H.261 standard, designed for video phones and video conferences. It's sometimes referred to as Px64 where the "64" refers to the 64K bit-per-second bandwidth granularity of digital phone lines offered by ISDN, while "P" can vary from 1 to 32 for low- to high-quality video.

An aside about video phones: I hope society is ready to deal with this idea. Besides having to button your shirt and comb your hair before answering the phone (Murphy says it will only be a "junk call" from some insurance salesman), I imagine problems like crank or obscene phone calls will reach new depths. Oh, well. Technology marches on and if it can be done, it must be done, right?



DCT TIME

All these standards share common technical underpinnings. The key difference between JPEG and MPEG or H.261 is the former deals with still pictures while the latter two handle motion pictures. In other words, still pictures are only compressed spatially [a single frame at a time or intraframe) while motion pictures are compressed both spatially and temporally (across frames or interframe).

The spatial compression is common to both, so let me start there. Essentially, compressing or "encoding" an image is done using a three-step process.

First, a block of pixels (typically 8 x 8 or 16 x 16) is operated on with a Discrete Cosine Transform (DCT), which effectively converts that portion of the image from the spatial domain into the frequency domain as shown in Figure la. Conceptually, the low-frequency components of the signal are shifted to the upper left corner and the high-frequency components to the lower right. Thus, an ordered frequency spectrum is generated when the block is scanned in a zig-zag fashion (see Figure 1 b).

Second, the frequency spectrum is "quantized," which simply means the signal level is quantified with a certain degree of resolution (quantization step size) as shown in Figure lc. For instance, a signal that varies from 0 to 255 (8 bits) could be quantized in steps of 128 (1 bit), 64 (2 bits), 32 (3 bits), and so forth.

Most importantly, the step size used can and should be different for each frequency component. The key point behind the DCT scheme is it relies on the human eye's penchant for noticing low-frequency errors much more than high-frequency errors. Thus, the low-frequency components should be quantized with higher fidelity, or smaller step size. The final result of quantization is the reduction of all data in magnitude (e.g., 8 bits to 2 bits), and many coefficients become zero, especially high-frequency ones, thanks to a large step size (Figure 1d}.

The zig-zag order in which the frequency spectrum was generated in the first step exploits the fact that images feature spatial frequency locality; a pixel tends to look like those around it. This trait, along with the many zeros generated by quantization, is well suited to run-length encoding, which is performed on the output of the second step.

Third, the run-length-encoded quantized frequency spectrum (the output) is further coded using a statistical scheme known as the Huffman code. Briefly, this code exploits redundancies in the data by assigning shorter codes to the more frequent elements and longer codes to those less frequent. Statistical codes require that both the encoder and decoder have a table mapping each code to the real data it represents; the optimal coding varies depending on the data. Thus, the standards contain provisions for standard tables in both the encoder and decoder as well as a way for an encoder to send a custom table to a decoder.

One key point is the DCT and quantization steps are "lossy," meaning the original input cannot be completely regenerated by reversing the compressing and encoding steps. Understanding that "lossless" compression just wouldn't be feasible, the PEGs chose the "losses" to exploit the fallibilities of our visual system (what you can see is only what you get). Of course, the variable quantization step size is the ultimate bandwidth safety valve: just keep making the quantization more coarse until the bandwidth falls to that of your storage device [within reason-I suggest you don't try storing video on paper tape!).

A "Silicon Update" brownie point if you hack some JPEG software on your PC (it has been done). It may run rather slowly, but hey-it's an ideal response to those "skeptical others" who dare question why upgrading to a faster computer is really smarter than buying a new car.

Just remember, JPEG is only for still images. Doing what's called "Motion JPEG" in which a sequence of individually compressed frames are strung together is possible. It's a short-term hack that works, but the ultimate solution is to compress across frames (i.e., MPEG).

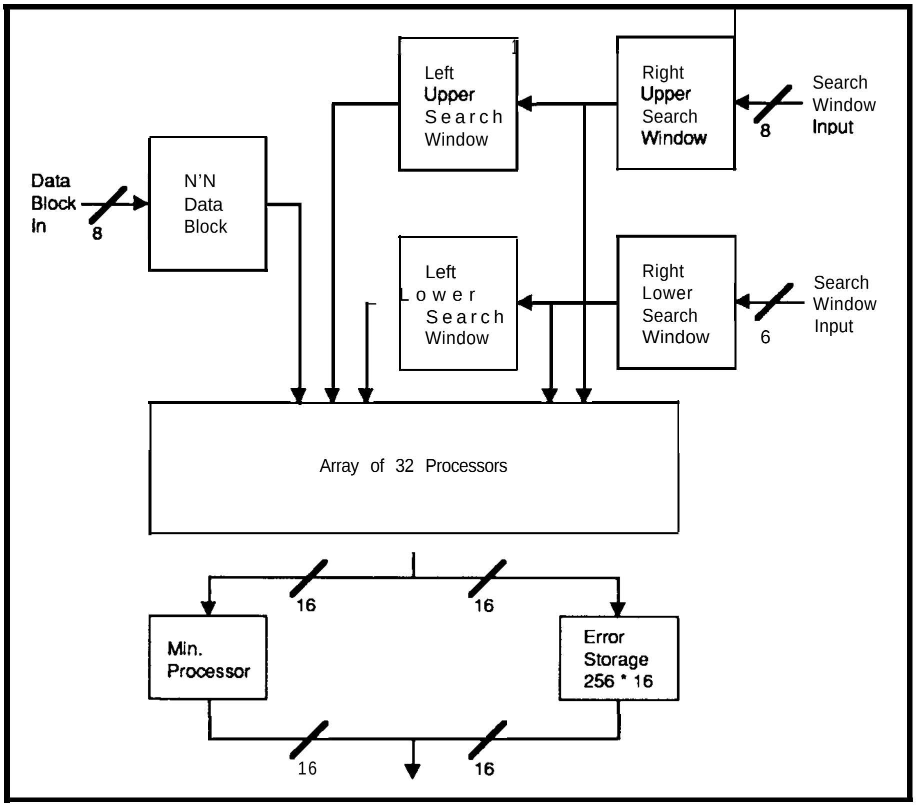


Figure 2—L64720 motion estimation processor block diagram.

WILD THING, YOU MOVE ME

You don't have to be a genius to understand the concept of frame differencing. Just as pixels have spatial locality [one pixel tends to look like others around it), they have temporal (time) locality as Well; a pixel in a given frame will tend to look the same as in preceding and following frames. Sports scenes are a great example. Imagine lining up the putt (or waiting for the windup or snap) in which seconds, or hundreds of frames, may transpire with little actually taking place, making you yearn for the "highlights at 1 l:OO." Instead of sending all these idle frames, you can send the equivalent of an NOP ("The pitcher is still checking the sign.") with only a few differences from the last frame ("The pitcher is still checking the sign, but he did spit.").

Static scenes compress well indeed, but what happens when the director shouts, "Action!" You discover some types of motion aren't that hard to deal with. One example is "panning," the smooth movement of the camera in a linear manner. This function is analogous to scrolling on a terminal. The simplest way [like JPEG) is to rewrite the entire screen. However, more sophisticated terminals include a programmable start address, so scrolling is as simple as specifying a new start address and transmitting only the newly visible information (a new line). Similarly, an MPEG-like motion compressor can just specify the equivalent of a new camera angle relative to the previous frame without transmitting the entire new frame.

If static or panning scenes are the good news, the bad news is random motion (i.e., after the snap) and zooming, not to mention complete discontinuities accompanying scene changes. Here, the technology gets pushed to come up with motion estimation schemes. Basically, the process requires high-speed comparisons between each block of pixels with

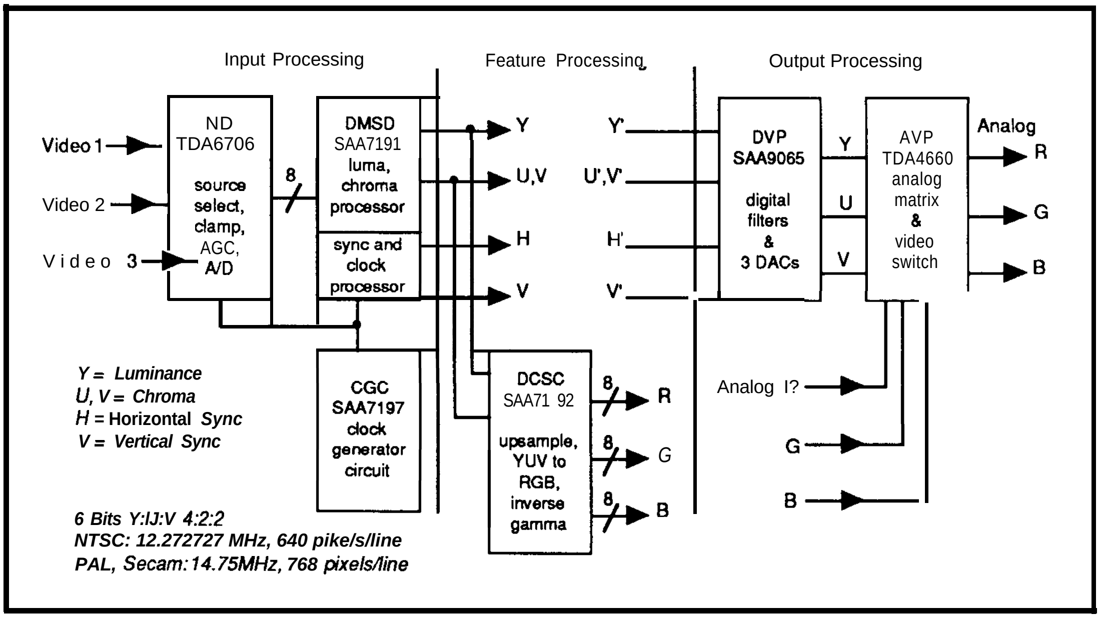


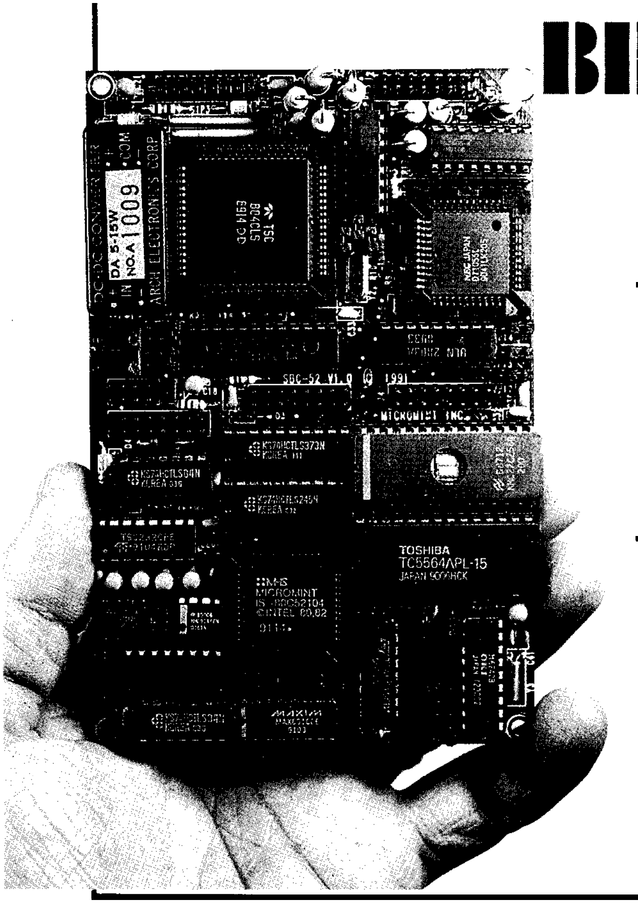
Figure 3—*Signetics video encoder and decoder.*

all the other blocks of pixels (the "search window") that surrounds them!

As you can imagine, doing this in real time [i.e., 30 frames per second) calls for some pretty heavy duty hardware. Check out the LSI Logic L64720 Motion Estimation Processor (see Figure 2), one of the leaders in JPEG and MPEG chips. Notice the little block labeled "Array of 32 Processors"!

In fact, while the DCT, quantization, and Huffman coding [i.e., JPEG) may require a few MOPS each, motion estimation (i.e., MPEG) ups the ante into the rarified atmosphere of BOPS! Fortunately, the silicon wizards are always willing to rise to a challenge.

One contributing complexity factor is that the MPEG encoder must also contain an decoder in order to perform motion estimation. Thus, the computational complexity of MPEG encode and decode is asymmetric; transmitting (encoding) is much harder than receiving (decoding). JPEG is much more symmetric because it doesn't deal with motion, which has key implications for applications that may want to author (transmit) multimedia, to read (receive] multimedia, or both.



BRUTE-52 Maximum Power Minimum Space

Micromint's BRUTE-52 is the ultimate compact controller. One look at the list of features will tell you that this full-featured controller has the power to crush your most demanding applications:

- . CMOS 80C52/80C32
- . Three 16-bit counter timers
- . 11.0592 MHz System Clock
- • Hardware Watchdog Timer
- . Hardware Clock-calendar
- • Optoisolated Serial Communications RS-232 or RS-485! 300-9600 bps!
- . Optoisolated Serial Printer Port, RS-232 1 50-9600 bps
- • 5V-only Operation
- . Up to 56 Kbytes RAM and/or EPROM
- . 1 Kbit EEPROM
- . 12-bit parallel TTL I/O
- . 8-bits buffered high-voltage. high-current outputs
- • 8-bits optoisolated non-polarized DC inputs
- • 12-bit plus sign analog-to-digital converter 8 channels! 60 Samples/second! 1.2 mV resolution!
- . 12-bit digital-to-analog converter 2 channels! 1.2 mV resolution! Selectable ranges!
- • Only 3.5 x 5.3 Inches!
- • Operates at C-70°C
- . Consumes only 100–200 mA (depending on configuration)
- • Use networked or stand-alone
- . DEMO/Diagnostic ROM

BRUTE-52 offers you all these features at only $459! ($379/100 quantity OEM) We also have a starter system for $289. When you add in Micromint's renowned quality, sevices, and support, you won't find a better value in compact control.

To order BRUTE-52, or for more information, contact:

Micromint, Inc.

4 Park Street • Vernon, CT 06066
Phone (203) 87 1-6 770 • FAX (203) 872-2204

#16

Today, the spectrum of alternatives spans the conceivable thickness of your wallets; everything from the software-only JPEG to zillion-transistor chip sets. Anyway, it's only a matter of time before it all ends up in a $5 chip.

ABANDON THE PAST?

Despite the inevitability of Superboxes, I'm not quite ready to start heaving my CDs, cassettes, and videocassettes. Heck, I've still got LPs!

In the case of video, what's clear is you and I are going to have to coexist with NTSC for some time. Thus, the need for adapter circuits between the old analog realm and the new digital one shouldn't be overlooked.

Fortunately, Philips/Signetics and others are responding with chips that bridge the gap between existing analog I/O and the emerging digital wonderchips. An example shown in Figure 3 illustrates how the analog interface chips surround the digital feature processor.

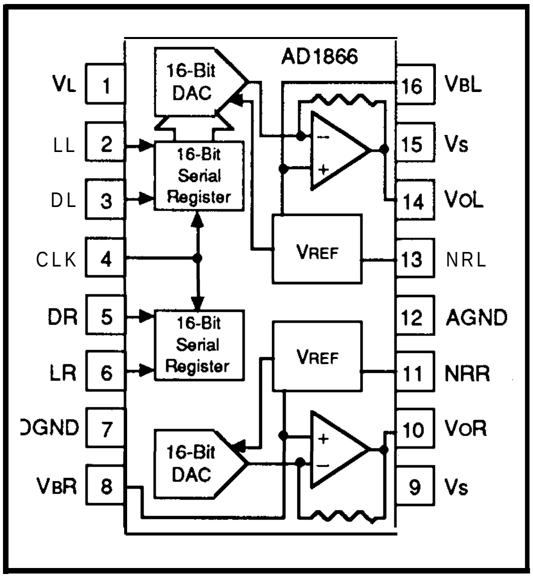


Figure 4—*AD1866 block diagram.*

Meanwhile, don't forget that multimedia includes sound as well as video. Thus, keep your eye out for digital audio chips like the Analog Devices AD1866 +5-volt-only dual-channel (stereo) 16-bit PCM audio DAC (see Figure 4).

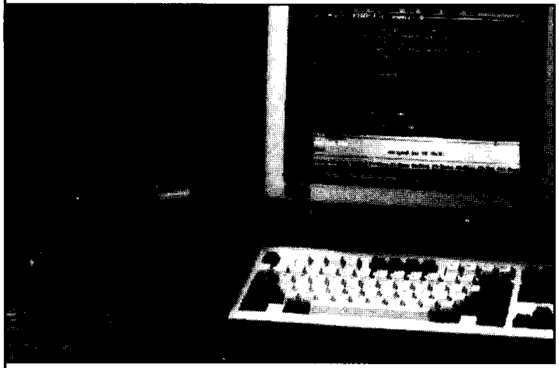
AND THE WINNER IS...

The fact is, all the chips and standards I've mentioned are sure to do well. But focusing only on these implementation technologies without considering the fundamental changes required to put all this stuff to use is like evaluating a few trees when constructing a forest's ecological model; most definitely a mistake.

Stepping back, assume that your PC will be able to deliver full-motion A/V-now what? Where will you get A/V to play, or even more critical, how will you create your own A/V? How will the A/V interact with the computer software? After all, multimedia has got to do more than just make your PC act like a TV or stereo. Indeed, with the increasing reliance on A/V for criminal trials, television news, and so forth, how will you know whether a "clip" is real or the product of some depraved hacker's sense of humor? If I "grab" some A/V off the airwaves and put it on a disk without

ROM Borland or Microsoft C

Complete 80x86 ROM development kit!



*Place your Microsoft or Borland C code in to ROM with the **C_thru_ROM** complete ROM development kit.*

Whether you use Borland C or Microsoft C to develop for ROM, **C_thru** ROM will help speed your projects to completion This complete ROM development package is the only product that gives you all the tools you need for easy development of 80x86 ROM code on your PC. **C_thru_ROM** includes: ROMable startup code; floating point support; a ROMable library that includes *printf, malloc, free,* etc.; and a full 80x86 locator that outputs binary, Intel OMF, and hex formats. You can debug with your Turbo debugger or use Datalight's remote debugger. Either debugger will work through an available UART or ROM socket UARTs. This comprehensive ROM development package is available for only $495. Call, write or fax today for your free demo disk and full details about **C_thru_ROM.**

Free Demo Disk! Call Today Toll-Free l-800-221-6630

Datalight

17455 68th Ave NE, Suite 304, Bothell, WA 98011, USA • 206-486-8086 • fax 206-486-0253

#169

the expressed written permission of the owner, will I get a visit from the intellectual property police?

I don't know the answers, but we'll find out soon because in my opinion the first "viable" multimedia product is now available. And the winner is.. . Apple and QuickTime.

The day job hearkens, so I can't do it justice. All I'll say is that the Macintosh, and QuickTime, is way ahead of everyone else. Consider the following:

•QuickTime "movies" are just another data type to the Mac. Indeed, my understanding is that existing Mac application software already works with QuickTime. If your program can cut and paste text or a PICT, it can already cut and paste a movie.

*The details of A/V compression are handled in a BIOS-like manner "underneath" and decoupled from QuickTime. For now, QuickTime relies on software compression of various types; JPEG for still images and Apple proprietary schemes for motion, animation, and audio. The bad news is that it is a little pokey, meaning the video resolution and frame rate is limited. The good news is it runs on existing Macs (at least the faster ones] without having to add any new hardware. The important news is that QuickTime can "transparently" adapt to evolution in the A/V compression arena [e.g., hardware motion JPEG to MPEG to MPEG 2).

*They've thought out a lot of the details including a standard "player" interface (i.e., play button, rewind button, etc.) and compatibility with the heathen masses (Windows users will be able to at least play QuickTime movies]. One contribution from the no-doubt well-staffed Apple legal department is that each "frame" has a field for copyright info.

•Watch the alliances with A/V leaders like Sony. I suspect a "MacTV" is right around the corner.

Ironically, QuickTime is the winner because, as Apple says, "It isn't multimedia," that meaningless gobbledygook of acronyms, "it's Macintosh." ❑

Tom Cantrell holds a B.S. and an M.B.A. from UCLA. He owns and operates Microfuture Inc., and has been in Silicon Valley for ten years working on chip, board, and system design and marketing.

CONTACT

LSI Logic Corp.
155 1 McCarthy Blvd.
Milpitas, CA 95035
(408) 433-8000
Fax: (408) 434-6457

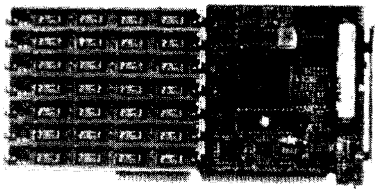
Philips/Signetics Company
811 East Arques Ave.
Sunnyvale, CA 94088-3409
(800) 227-1817

Analog Devices, Inc.
18 1 Ballardvale St.
Wilmington, MA 01887
(617) 937-1428
(617) 821-4273

I R S

422 Very Useful
423 Moderately Useful
424 Not Useful

ROMDISK™



High Performance
Multimegabyte Disk Emulators

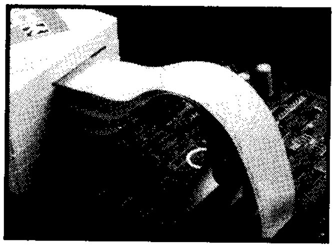
NEW MODELS / LOWER PRICES

- Floppy Drive and multimegabyte emulators for ISA bus computers
- 180K to 14 MB capacities
- EPROM, Flash or SRAM technologies
- Autobooting, Single or Dual disk emulation under PC or MS DOS
- List prices from $195

CURTIS, INC.
418 W. Country Road D . St. Paul, MN 55112
612/631-9512 FAX 612/631-9508
PC DOS is a trademark of IBM, MS DOS is a trademark of Microscoft

#17

ROM-IT
EPROM EMULATION SYSTEM



NEW
4-MEGABIT
VERSION

■ Emulates up to 8 4-Megabit EPROMS with one control card.

■ Downloads 2-Megabit programs in less than 23 seconds.

■ Allows you to examine and modify individual bytes or blocks.

■ Accepts Intel Hex, Motorola S-Record and Binary files.

■ Software available for IBM PC and compatibles and Macintosh systems.

■ Base 27256 EPROM System $395.00 Other configurations available.

ORDER TODAY--IT'S EASY
CALL OR FAX FOR MORE INFORMATION

Incredible Technologies, Inc.
(708) 437-2433
(708) 437-2473 Fax
VISA now accepted.

#171

Use a Watchdog to Keep Your Controller in Line

PRACTICAL ALGORITHMS

John Dybowski

A watchdog can be an invaluable aid in keeping a system completely reliable, but only when properly implemented. John gives some hints for what to do and what not to do.

Examining the terminology of our craft with some attention to detail is at times informative. Sometimes the descriptive terms become commonplace and overused, obscuring their true meaning. We all know about embedded controllers but what does this term really mean to us?

To embed means to deposit, locate, implant, or enclose closely by a surrounding mass. As designers, we know that these definitions strike close to the mark because our contrivances are often concealed by the mechanisms they are intended to control or monitor. This description, far from being merely a matter of semantics, denotes the differentiating factor between general-purpose computing devices and dedicated embedded controllers.

If we take this meaning to be indicative of the types of environments that our embedded controllers exist in, the need for reliable operation becomes quite clear because should these devices prove to be faulty, accessing the malfunctioning controller for repair can in of itself prove to be a problem. Furthermore, many of the applications themselves may be critical in nature. Embedded controllers are commonly used in such applications as access control systems, automobile controllers, and controllers for industrial machinery to name a few. Imagine a malfunctioning access controller that decides flinging all the doors open is a good idea. Worse still, what if a controller did not respond properly in an emergency situation and did not fling the doors open!

WATCHDOGGING

Most experienced engineers are well aware that, even using dubious design practices, one of anything can be made to work on the lab bench. This realm is in fact that of the hobbyist and is the point farthest from the actual operating environment of commercial embedded controllers (As is the projected quantity, hopefully!).

Even if the hardware and firmware are well designed to begin with, the problems associated with external disturbances still have to be addressed. Now I know this area is difficult to assess because basically it involves unknown elements that may be unique to the prevailing conditions present only in certain installations. Although the causes vary, the result is often the loss of the controlled execution of software, which can send the microcontroller into an indefinite period of seemingly random operation [the processor goes off the rails and starts running off in the fuzz). This situation is sometimes the result of electrical transients induced by power fluctuations (perhaps aggravated by a deficient reset circuit), static discharges, or electrical disturbances caused by machinery or close lightning strikes. A direct lightning strike will make loss of software control look like a walk in the park. All things are relative, aren't they?

This type of interruption of normal operation can damage essential data elements stored in nonvolatile RAM, causing ongoing problems as well. After having covered the topic of coding for this eventuality in my last article, I'll not belabor this point any further.

Ever since the first controllers misbehaved, designers have been contriving schemes to keep them in line. Although many circuit implementations have been tried over the years to resolve this problem, the idea consistently used is based on the principal of the retriggerable one-shot.

The premise behind this scheme is a properly executing program will strobe the watchdog at critical check points so the time-out condition of the one-shot is never reached during normal program execution. Should this activity cease for the duration of the time-out period, the watchdog insists the processor reset and restarts the system, effectively restoring controlled software execution. In other words, as long as the watchdog gets stroked everything is OK, otherwise it hits the processor with what amounts to a two by four over the head, gaining attention by yanking the reset line. Crude but effective. Yes folks, brute force has its place in controller work.

For simple applications, pulsing the watchdog from various points in the mainline code may be accurate. Alternatively, the pulse can be emitted from a timer interrupt routine. However, making the issuance of this pulse contingent on the operation of both a foreground and an interrupt-driven process is much better. The problem with the mainline approach is that the program may be running the foreground code just fine, but the interrupts may have become inoperative. On the other hand, the interrupts may continue to execute long after the foreground has fallen into a tight do-nothing loop.

Most programs generally perform certain functions on an intermittent basis, usually timing them with a timer interrupt as a time base. My personal preference when implementing these functions is to allow the interrupt handlers to do as little processing as possible. The approach I usually take is to run an interrupt-based idle timer that gets loaded in the mainline code and is decremented by the timer interrupt handler (counting stops when the timer equals zero). Often, I run the timer interrupt at 1 ms and use a reload value of 20 for the idle timer. A solid 20-ms time base makes counting much longer delays easier without imposing any additional burden on the timer interrupt routine.

I make this adjustment by allocating the required number of timers and nesting each in successively deeper layers of the foreground process code. A timer is reloaded every time it expires, then the program falls through, servicing the next lower layer along with an associated timer. I use this method to obtain very long delays with a minimum amount of overhead.

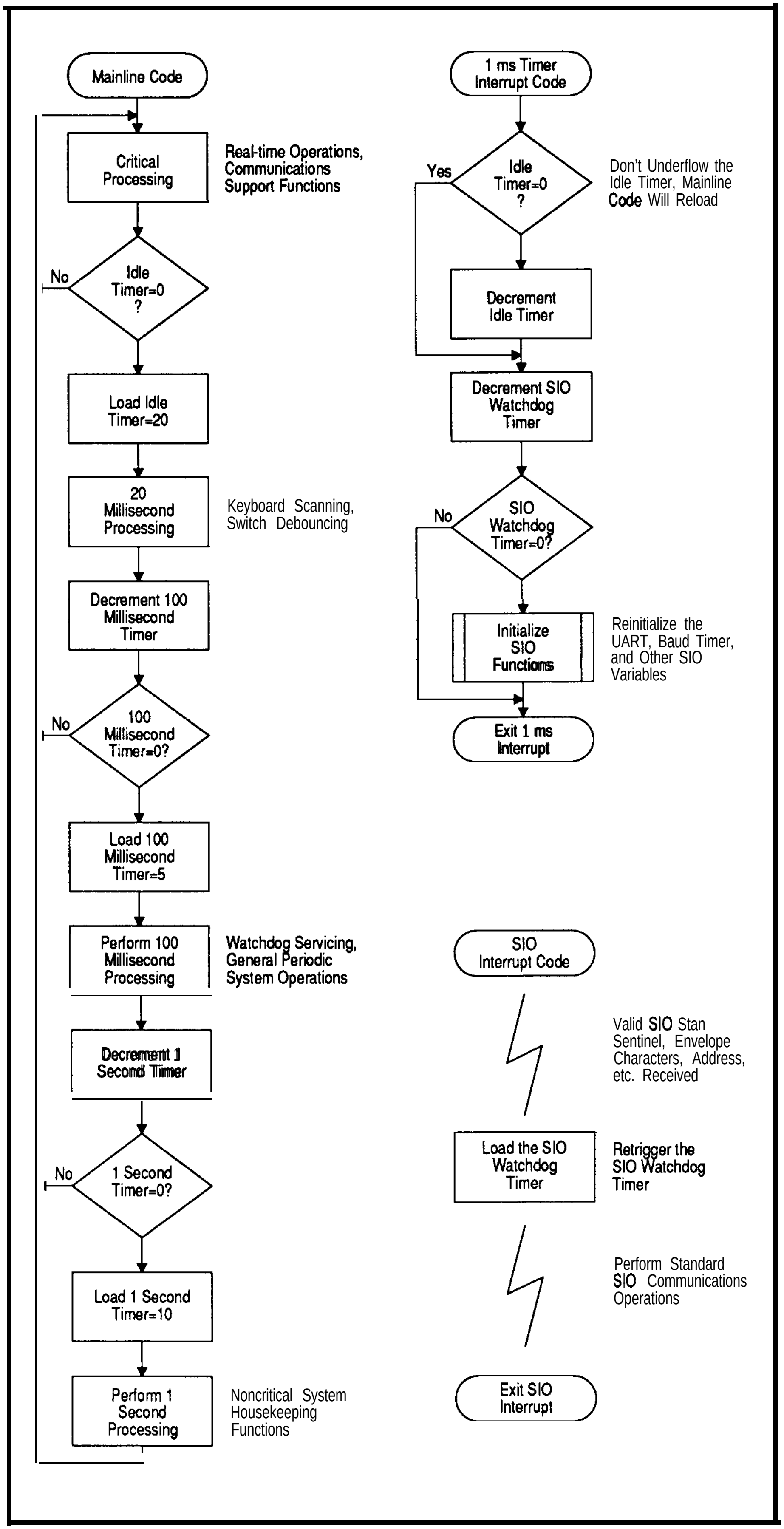

Mainline Code
Critical Processing
Real-time Operations, Communications Support Functions
No
Idle Timer=0 ?
Load Idle Timer=20
20 Millisecond Processing
Keyboard Scanning, Switch Debouncing
Decrement 100 Millisecond Timer
No
100 Millisecond Timer=0?
Load 100 Millisecond Timer=5
Perform 100 Millisecond Processing
Watchdog Servicing, General Periodic System Operations
Decrement 1 Second Timer
No
1 Second Timer=0?
Load 1 Second Timer=10
Perform 1 Second Processing
Noncritical System Housekeeping Functions
1 ms Timer Interrupt Code
Yes
Idle Timer=0 ?
Don't Underflow the Idle Timer, Mainline Code Will Reload
Decrement Idle Timer
Decrement SIO Watchdog Timer
No
SIO Watchdog Timer=0?
Initialize SIO Functions
Reinitialize the UART, Baud Timer, and Other SIO Variables
Exit 1 ms Interrupt
SIO Interrupt Code
Valid SIO Stan Sentinel, Envelope Characters, Address, etc. Received
Load the SIO Watchdog Timer
Retrigger the SIO Watchdog Timer
Perform Standard SIO Communications Operations
Exit SIO Interrupt

Figure 1—*The approach taken here is to run an interrupt-based idle timer that gets loaded in the mainline code and is decremented by he timer interrupt handler (counting slops when the timer equals zero).*

Figure 1 illustrates how such code is structured.

Being dependent on the proper operation of the mainline application program as well as the timer interrupt handler, this arrangement provides just the place to put the watchdog service code. Of course, you still need to provide the stimulus to the watchdog if you leave the mainline for any length of time, something that is especially true when doing the usual power-on diagnostics, such as the PROM and RAM test. In this situation, the pulses can be issued directly from the diagnostic routines because, in some cases, they may take a long time to execute. Another possible scenario exists if the mainline is exited during normal execution. Say I am to accept a download from the host and I elect to perform this function from a dedicated piece of code. Here, I may also use the idle timer to an advantage not only as a reference for the watchdog, but also to provide accurate ticks that can be used to count an abort interval should the host drops off-line.

The principal of the system watchdog can be extended to other ancillary functions as well. One area that can easily be protected using this scheme is the SIO. However, the feasibility of this usage does depend on the type of communications the system incorporates. Good results can be obtained when using host-driven protocols, especially polled protocols. Again, the idea of the retriggerable one-shot is employed where a timer is decremented in the timer interrupt routine. The SIO ISR reloads this timer at some point sufficiently deep in the protocol, ensuring that some intelligible communications activity is being detected. That the reload does not occur merely on an SIO interrupt is important because this condition could occur while receiving nothing but gibberish.

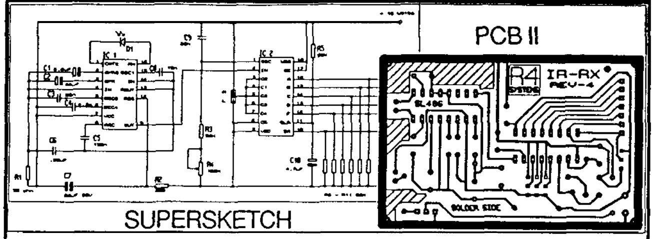
On expiration of the SIO watchdog timer, the timer interrupt invokes the SIO initialization routine that reconfigures the UART, baud rate timer, and so forth. This procedure could also be extended to determine the appropriate communications parameters, eliminating the need for manual configuration. In this case, the SIO initialization could be tried with different baud settings, parities, and word lengths each time until valid communications were established.

WATCHDOG CIRCUITS

Many chip manufacturers have recognized the need for a reliable microprocessor watchdog, and many ICs are now available that perform these functions. The development of reliable integrated watchdog circuits relegates the homegrown techniques consisting of counters, one-shots, and discrete implementations to the realm of relics. Being familiar with the Dallas DS1232, I'd like to discuss this part briefly.

The first problem addressed by the DS1232 is that of providing a reliable processor reset signal. (The DS1232 has both active-high and active-low reset outputs.) This signal is produced using a precision temperature-compensated reference and comparator circuit

EASY TO USE CAD TOOLS



Easy to Use PC Software - ICON Based
Ultra Fast Performance - Mouse Driven
Complete With Advanced Editing Tools
Output to Printers, Plotters & Lasers

PCB BOARD DESIGN
Full range of products available from entry level through to auto routing.

SCHEMATIC CAPTURE
A family of products that feature modern graphical user interface & an intelligent diagram editor.

PCB II or SUPERSKETCH $149 US !!!

R4 SYSTEMS Inc.
P.O.Box 451
West Hill, Ontario
Canada MI E 4Y9
(416) 439-9302

Free Evaluation Package
Write or Call Today

Download DEMO from BBS at 416 289-4554 (2400/8/N/l)

ROMboy

Low Cost, Reliable, High Quality ROM Emulation from the Creators of PROMICE

- Emulation of up to 1 Mbit ROMs (4 Mbit-soon)
- Supports any ROM type (DIP or PLCC)
- Supports high speed ROMs
- Supports any word size up to 2048 bits wide
- Host software for DOS, Unix, Mac, VMS
- Battery backed memory
- Intelligent micro-controller based unit
- SMT CMOS layout for robustness
- Tiny size: 2.5" x 3.8" x 0.9"
- Unbelievably low price!

ROMboy...

from the authority in firmware development tools

Orders: (614)471-l 113
West: (415)750-0219
FAX: (614)475-6871

Grammar
Engine
Inc.

#17

Lkting 1-A *watchdog can be used to keep a serial port from losing ifs power-up configuration.*

```
;POWER UP ENTRY
        MOV     IE,#00000000B       ;DISABLE ALL INTERRUPTS

        MOV     P0,#11111111B
        MOV     P1,#11111111B
        MOV     P2,#11111111B
        MOV     P3,#11111111B       :SET PORTS TO IDLE STATE

        MOV     IP,#00000000B       :SET INTERRUPT PRIORITY

        MOV     PCON,#00000000B     ;NORMAL POWER, NORMAL BAUD RATE
        MOV     TCON,#00000000B     ;TIMERS OFF
        MOV     TMOD,#00100001B     ;TIMER 0: NONGATED MODE 1
                                    ;TIMER 1: NONGATED MODE 2

        MOV     PSW,#00000000       ;SELECT REGISTER BANK 0

        MOV     SP,#7               ;SETUP SYSTEM STACK

        CALL    RESET_INT
        CALL    RESET_INT           ;REARM INTERRUPT PRIORITY LOGIC

        MOV     SCON,#01010000B     ;SET UART TO 8 BITS, RECEIVE ON
        MOV     A,#0FDH             ;9600 BAUD
        MOV     TH1,A
        MOV     TL1,A               ;LOAD TIMER REGISTERS
        SETB    TR1                 ;BAUD TIMER ON

        JMP     MAIN
RESET_INT:
        RETI
```

that monitors the status of the Vcc. When an out-of-tolerance condition occurs (which may be set at 4.5 volts or 4.75 volts), reset is forced to its active state. When Vcc returns to an in-tolerance condition, reset is kept active for 250 ms to allow the power supply and processor to stabilize. This status remedies the problems associated with brownouts and power dips in which simple RC circuits just don't work. [The DS1232 also provides for a push-button reset.)

The main capability of the DS1232 is the function of a watchdog timer. This usage is established using an internal timer (which requires no external components) that forces the reset to an active state if the strobe input does not see an active-low transition prior to time-out. This time-out interval can be set to 150 ms, 600 ms, or 1.2 seconds.

Unlike a regular one-shot that needs an initial pulse to start up, the DS1232 watchdog operates as a free-running timer that will continuously reset the processor in the absence of strobe pulses. On expiration of the watchdog timer, the reset interval begins and ensures a reset pulse of adequate duration before the watchdog timer starts free-wheeling again.

SOFTWARE RESET CODE

The power-up code is the part of the program that gets control immediately following the processor's emergence from reset. Mundane and somewhat tedious functions related to system initialization are performed here, configuring the processor and peripherals for subsequent operation. Microprocessors and particularly microcontrollers, especially those that integrate many peripheral functions on-chip, usually have a number of parameters set to default conditions as a result of a hardware reset. Often, these defaults are just what will be required, and the temptation is to leave well enough alone and proceed with setting up only those functions that differ from their default settings.

This approach can cause problems because if the code is entered by a means other than from a hardware reset, things may not be as you expect. For example, using the power-on entry point as a target for certain error conditions that by their nature inhibit further processing during normal program execution is not unreasonable. Having reached an impasse and being unable to resolve a system conflict, the process code may simply do a jump to reset with the intention that the power-on sequence may be able to sort out the mess and either restart the system, issue a distress call to the host computer, or take some other corrective action. The reset point may also be reached because of errant program execution. Perhaps the processor got lost and ran through memory, wrapping around back to the reset entry point. Finally, I have seen a communications protocol that actually defined an embedded reset command as part of the protocol layer!

What all these possible problems amount to is that assuming a set of default conditions on entry into the software reset sequence is not a good idea. All system functions should be defined explicitly in this piece of code. Although simple in principal, this approach requires some careful thought. For example, say you are forced to implement the screwy protocol I described, where resetting the system must be done directly from the interrupt level. For the moment, say this function will run on an 803 1 processor. Now, you may think that if the power-on code sets all the registers, ports, and other processor resources to the desired state, all will be well. This assumption is not at all the case and you end up instead with a nonfunctioning system.

What may not be immediately obvious is the SIO (and any lower priority interrupts as well) will not be functional following this maneuver. What happens is the in-service bit for the SIO interrupt level [which is only cleared on the execution of a RETI instruction) remains set, blocking any interrupts of equal or lesser priority. The bad news is the in-service bits are buried in the innards of the 803 1 and cannot be manipulated directly. The good news is there does exist a way to work around this aspect. The problem can be remedied by performing a call to a dummy routine consisting of a single RETI instruction, which has the effect of clearing the lowest priority in-service bit in effect at the time. (If no in-service bits are set, no problems are caused.) Do two such calls in sequence and you're covered if things really have gotten botched. Of course few of us would be so foolish as to jump voluntarily to reset from an interrupt level, but the fact remains that such an event could be the result of a system anomaly.

CRASHPROOF CONTROLLERS

The watchdog circuits that I've described up to this point are appendages that can be applied to general-purpose processors and controllers to enhance their reliability. The need for such features is now becoming widely recognized and these circuits have appeared as integral components of some of the newer processing elements on the market. Therefore, a look at a couple of these implementations to see how they surpass the capabilities attainable using the add-on approach is instructive.

The Dallas DS5000 [and functionally equivalent DS2250) microcontrollers have an apparent similarity to the popular 8031 device. However, several features have been added to help ensure the orderly execution of the application software in the face of harsh electrical environments. Specifically, timed access control, a watchdog timer, and a power-fail interrupt have been built into these parts to help provide control and recovery under difficult operating conditions.

The timed access feature is used to control access to the critical configuration and control bits in the DS5000 special function registers. These protected bits, which include the enable watchdog timer, the reset watchdog timer, and stop mode, may only be written through the execution of a specific instruction sequence involving the timed access register.

In order to modify any of the protected bits, a pattern of two bytes consisting of an AAh and 55h must be written to the timed access register within two cycles of each other. After this sequence is performed, the protected bits may be modified within four cycles of the second write (the 55h). If either of these timing constraints are violated, then the timed access is reset. The timed access function can protect against the possibility of inadvertent write operations of a critical bit. Of course, it cannot protect against the case of inadvertently entering a loop that contains the correct instruction steps, but it does greatly reduce the chances of unintentionally affecting these system areas. This controlled access means of affecting the watchdog circuit emphasizes the need for limiting the possibility of unintentionally affecting this critical system function. When using an external watchdog circuit, don't tie the strobe pin to an extra chip-select line that you may have available. In such an arrangement, an errantly executing program can defeat the watchdog far too easily by falling into a loop that will inadvertently issue strobes to the watchdog.

The DS5000 also contains status bits that indicate the cause of the reset. These embedded bits, along with user-defined indicators, may be interrogated on power *on* to help sort out the mess I alluded to earlier.

The PIC microcontrollers from Microchip also have built-in watchdog timer circuits. These small, low-cost controllers serve best in relatively simple applications. What's interesting here is the watchdog timer is free running and uses an independent on-chip RC oscillator. Not requiring any external components, the watchdog timer will run even if the CPU clock is stopped (which could be the result of executing a SLEEP instruction). Furthermore, the PIC uses a Harvard architecture that separates the program and data areas, and the possibility of getting out of sync or executing data as code does not exist because all instructions are in one-word lengths.

With a capacity for intelligent operation, a low cost, and presumed reliability, the PIC presents an intriguing potential as a sophisticated watchdog controller in a system hosted by a more powerful controller. Having decision-making capabilities, the PIC can be used not only to detect the master controller's activity, but to qualify this activity and make a determination whether or not the system master is operating in an appropriate manner. If a problem exists, the PIC can try restarting the system using the proverbial two by four. If that attempt fails to remedy the situation, the master controller can be switched out and a backup system activated. This arrangement would be epitomized in a system running under the control of a computational power house with the aid of the street smart PIC to keep it in line. Or is it really running under control of the PIC?

The bottom line remains the same: Stuff Happens (putting it delicately). This fact exists because of the action of forces that we may not fully appreciate, influenced by the vagaries of the moment. Armed with this knowledge, we shape our thinking accordingly. As good engineers, we trust nothing and craft our designs conservatively and carefully. ❑

John Dybowski has been involved in the design and manufacture of hardware and software for industrial ***data collection*** and *communications equipment.*

SOURCES

Dallas Semiconductor Corp.
440 1 South Beltwood Pkwy.
Dallas, TX 75244-3292
(214) 450-0448
Fax: (214) 450-0470

Microchip Technology, Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
(602) 963-7373
Fax: (602) 345-3390

I R S

425 Very Useful
426 Moderately Useful
427 Not Useful

CONNECTIME

conducted by Ken Davidson

The Circuit Cellar BBS
300/1200/2400 bps, 24 hours/7 days a week
(203) 871-1988—Four incoming lines
Vernon, Connecticut

In this issue's ConnecTime, I'm going to wander away from computers slightly with one of the threads. We had a very informative discussion involving photography, shutters, and strobe f/ashes I thought you might find interesting. First, though, I'll start with a topic many of us have encountered in the past at least once: choosing a real-time clock for a microcontroller.

Msg#:49982
From: ROBERT SCHULTZ To: ALL USERS

Does anyone use or know of a product that will keep time for a microcontroller? I need date and current time of day; any date format will do. I have room in the design to access it serially or via the bus. I have looked into Dallas Semi's products. Is there anything else?

Msg#:50012
From: AL DORMAN To: ROBERT SCHULTZ

Try the RTC 58321 by SaRonix, (415) 856-6900, (800) 227-8974. I did figure out how to talk to the Dallas serial chip a while ago; it should be posted here somewhere.

Msg#:50170
From: MICHAEL MILLARD To: ROBERT SCHULTZ

I just finished a project that needed a real-time clock and I selected the National Semiconductor MM58274C. This chip is pretty standard in a lot of the designs I have seen over the years so it must work OK. I had very little difficulty getting it up and running. It's a 4-bit addressed device so it was real easy to hang it off one of the project's many 8255 PPIs.

It's also simple to back up the contents of the clock RAM with a small 3V battery. There is some extra circuitry, however [a couple transistors and a diode or two). The part uses a 32-kHz crystal for a timebase. Not to mention...everybody stocks them. They are here to stay!

Standby current is just 10 μA at 2.2 volts;16-pin DIP or 20 pin PLCC. Same pinout as National's MM58 174A and MM58274B. You can get the part for anywhere from $2 to $6 each.

National publishes a databook on these things: National Semiconductor -Advanced Peripherals, Real Time Clock Handbook. It is available from Digi-Key, phone (800) DIGI-KEY, part# 9129B.

Msg#:50096
From: ED NISLEY To: ROBERT SCHULTZ

The goofy little Oki MSM6242 works OK, but you ought to leave room for a variable capacitor on the board because the out-of-the-box accuracy isn't too good. It's a bus-connected device with a four-bit interface...other than remembering to start the fool thing the first time you "talk" to it, there's nothing tricky about making it play.

The C and assembler code I did for the thing back in the Furnace Firmware project is still spinning around on the BBS if that's of any help.

Msg#:50217
From: FRANK KUECHMANN To: ED NISLEY

Er, Ed, isn't the "out-of-the-box" accuracy of the 6242 clock chip essentially determined by the caps you use in the oscillator rather than by the 6242 chip itself?

For accuracy (or lack thereof] you can truly and justifiably pin on the manufacturer, there's a pin-compatible clock chip made by Epson with an on-board crystal. No way to tune it even if you want to. Available in 10 PPM and 50 PPM max error versions. I think it has a 72421 part number. Digi-Key has had 'em in the recent past.

Msg#:50356
From: ED NISLEY To: FRANK KUECHMANN

Well, it's crystal controlled, but the specs indicate that "even so" you're in the few tens-of-parts-per-million range. It's got a test mode that you can use to twiddle the thing to the right value; methinks.

The basic accuracy should be a few seconds per month, but the two I've used have been off by a _lot_ in a few weeks without adjustments. One of these days I'll have to set one up and let it run for a few months while recording the time once a day or so, then plot deviation versus time and temperature and see what comes out. Might make for an intersting article.. .

Msg#:50446
From: FRANK KUECHMANN To: ED NISLEY

I'd guess variations and inconsistencies in the 6242's timekeeping are more likely to be found in places other than the clock chip itself: Things like capacitors, board layout, temperature variations, and so forth would seem more important than the chip itself. I usually just use the Epson 72421 with the on-board crystal and have had no problems with erratic timekeeping.

Msg#:50474
From: ED NISLEY To: FRANK KUECHMANN

All my times are in "house temperature" environments, so [in principle) there shouldn't be too much thermal effect...and I don't believe that either. Gad, one more variable to explore.. ..

Msg#:50439
From: JEFF SUTHERLAND To: ED NISLEY

I've had a bit of experience with that Oki RTC chip in 8088 and 8031 circuits. It is VERY noise sensitive! Not routing signals properly will cause that thing to gain time like nobody's business. Note that the crystal has to be bypassed to Vcc, NOT ground. This will enhance noise immunity. Keep data and address lines away from the crystal, and if you're laying out a PC board, run a heavy ground trace all around the crystal parts. Run no signals under the crystal area on the other side of the board or on inner layers unless they're on the other side of a ground plane. With a variable cap (5-20 pF) in parallel with one of the crystal bypass caps I've been able to get my thermostat computer that runs my boiler to hold to within about 2 seconds/month. Since the chip only draws about 2 µA in standby mode, it's great for something that has to be battery backed-up. It'll run off a coin cell for a year or more easily.

While photography may have nothing to do with *computers in the strictest sense, the world of stop-motion photography deals with sometimes sophisticated electronics and timings in the milliseconds or even microseconds.*

Msg#:50664
From: DAVID MACDONALD To: ALL USERS

I need a fast responding (presumably electromechanical] camera shutter that can be placed in front of a single lens reflex camera lens and that will fully open in 2 ms or less following electronic triggering. Once open, several electronic flashes have to be fired, then the shutter has to close (after being open for a maximum of 1/60 sec.). The camera's own shutter would, of course, be open all the time.

Conventional camera shutters take 10 to 100 ms to open. Since I will be relying entirely on artificial light, the shutter need only be a disk that rotates away from the lens and not a complicated multileaf device. It seems to me that electromagnetic devices such as solenoids are inherently slow because of the time required to build the magnetic field. Does anyone have any experience with or ideas about this problem?

Msg#:50676
From: STEVE LANGER To: DAVID MACDONALD

There are things called Kerr cells, which are extremely fast electrochemical shutters based on partial depolarization of polarized light when it travels between electrostatically charged electrodes in a nitrobenzene dielectric. Not difficult to build, but can be had for some $$ already ready made. They are faster than anything you described. Need a special power supply and driving circuitry, though.

Msg#:50836
From: MIKE RAPP To: DAVID MACDONALD

I'll make several suggestions, all of which circumvent the need for a high-speed shutter.

1. Just lock the camera shutter open for the duration. Your message states that your only light will be artificial (the strobes?). If this is the case then the only image that you'll record is that produced by the strobes. You can have the shutter open for seconds, minutes, or even hours in the dark with no problem. This is how all those slick strobe photos are taken: turn out the lights, open the shutter, drop the marble toward the bowl of milk [or whatever), flash the strobe at the proper time [either time delay or sound activated), close the shutter. All this does assume you can lock the shutter open; that's what the "B" [bulb) setting is for. Older cameras need a lockable cable release, newer cameras can be controlled via an electric switch. You can even do this sort of thing in less than total darkness by combining a colored light source and a filter on the lense. Stopping way down can help, as can slower film.

2. If you absolutely, positively (sounds like a UPS ad) have to synchronize a shutter to a fast action then consider triggering from an earlier event (with a possible delay). Say you need to photograph a bullet passing by 50 ms after it was fired. Let's assume your shutter takes 16 ms to open. Start a 34-ms delay at the time the gun is fired. At the end of the delay you activate the shutter. Sixteen milliseconds later the shutter is fully open-just as the bullet goes by.

3. Another common technique is to turn everything on its head: have the camera control the experiment instead of being controlled. The normal socket for flash cables (X sync, I believe) gets its contacts closed at the exact time the shutter reaches full open. If circumstances allow, just use this to start the experiment and you'll be sure of the shutter being open in plenty of time.

If none of these ideas help then maybe you should tell us a little more about what you're trying to do. It may not

CONNECTIME

get any better replies but I would sure like to know why the above ideas wouldn't work! :-)

Msg#:50857
From: DAVID MACDONALD To: MIKE RAPP

Thanks for your very interesting and detailed reply. I will try to explain just what I'm up to, as I should have done right from the beginning. I want to photograph insects in flight. The general setup (indoors) will have an insect such as a bee or a butterfly flying out of a flight tunnel towards a light (its motivation) and the camera. Several strobes (40-µs duration approx.) will freeze the wing motion, which can exceed several hundred beats per second.

You are clearly very knowledgeable about photography, so you'll be well aware that at magnifications of 1:2 or 1:1, depth of field is very, very shallow. Couple that with an insect moving in an erratic fashion and at perhaps 2-5 mm per millisecond, and one has a challenging technical problem. I know of one photograher (Stephen Dalton) who overcame these difficulties and produced astounding pictures of insects, birds and other creatures in motion. He used an interrupted light beam to trigger the camera, and a fast homemade shutter placed over the lens of the SLR camera that could open fully 1/450 second after being triggered. All I know about his shutter is it used springs. I have, unfortunately, been unable to access his design.

I can see that a complementary approach is to use several light beam sensors to estimate the insect's velocity and predict when to fire the camera shutter. Nevertheless, it still holds that the faster opening one can make the shutter, the greater success rate one will have overall.

I've toyed a very little bit with a rotary solenoid and a cheap DC milliammeter, reasoning that both have a rotary action that would facilitate rotating a disk out from in front of the camera lens. The solenoids seems to actuate in about 20 ms; doubling its rated voltage certainly helps its speed considerably, but that is still far away from 2 milliseconds. The meter (undamped I think) seems to be even slower. Electromagnetic devices have always baffled me, so I don't really know how to proceed in this direction, or whether indeed it might ultimately be fruitful.

Not only must the shutter open quickly, it must also close rapidly so the total open time won't exceed about 1/60 second because there will be ambient artificial and perhaps natural light that could impress an image on the film if the shutter was held open any longer.

I will certainly welcome any and all suggestions.

Msg#:51130
From: MIKE RAPP To: DAVID MACDONALD

OK, I see your problem. I'm not yet ready to concede that you need a super fast shutter, though. Here's what I would try: Set up a dark background. A sheet of black velvet is what is most often recommended. Stop down as far as you can-f16, at least, f22 or f32 is better if your lens can. Choose a slow film-1 assume you want color so load up some Kodachrome 25 (ASA 25). Believe me, you won't have problems with ambient light at any normal shutter speeds indoors [heck, normal exposure for ASA 25 is 1/30 at f16 in bright sun!). I assume you have some way of detecting the approaching insects (how else would you fire strobes at the right time?). To me your biggest problem might be getting nice, even lighting on your little subjects so you get pleasing results. At close distances the angle between the camera and the light source can become very great, which results in severe cross lighting and shadows [that's why the Medical Nikkor lens has a circular strobe around the lens).

A few other thoughts: I'm not familiar with Dalton's work, but if he was working outside then he had different requirements; bright sky is far from a dark background :-). Second, be wary of any solutions that involve a moving slit or slot. These are not compatible with strobes since you'll just get a stripe of image on the film. That is why focal plane shutters have a maximum strobe flash speed [usually

MORE CIRCUIT CELLAR PROJECTS!

The Circuit Cellar Project File, Volume 1 has over 200 pages of new and expanded hands-on projects and tutorials.
The Computer Applications Journal's editors have chosen a dozen of the top projects from the Circuit Cellar Design Contest, independent submissions, and top-response articles to make a book with something for every interest!

now only $17.95! [includes domestic delivery*]
Order your copy today!
*$17.95 Visa, Mastercard, Check, or Money Order (US Funds Only)
(Add $2.00 for delivery to Canada or Mexico, $4.00 for delivery to other non-U.S. addresses)

The Circuit Cellar Project File. Volume1
4 Park Street, Vernon, CT 06066 • Tel: (203) 8752199 • Fox: (203) 872-2204

CONNECTIME

1/60 or 1/125}. At higher speeds they are really a slot moving from right to left across the film. Lastly, be prepared to waste some (perhaps a lot] of film. It's surprising how much film the pros go thru sometimes to get just a couple of good images (even when the subject is well lit and stationary), I'm not arguing in favor of film waste; it's just that in this case film might be cheaper than a high-tech shutter. You're going to be wasting film anyway because of the uncertainties in distance and lighting, no matter how fast your shutter might be.

Msg#:51166
From: ED NISLEY To: MIKE RAPP

Speaking of film waste.. .

"If your photographs don't look as good as those in National Geographic, there is a reason. Each year when National Geographic's 18 staff, 10 contract, and 35 freelance photographers take to the field, they take their pick from 900 cameras and 3000 lenses. They shoot 35,000 rolls of file a year or over 90,000 photographs per issue. And these are professionals."

From Illustrating Computer Documentation, William Horton, ISBN O-471-53845-0, $32.95. A good book if you _think_ you know how to write manuals and suchlike.. .

Msg#:51278
From: DAVID MACDONALD To: MIKE RAPP

Thanks for your further thoughts on the subject. I have intended to use Kodachrome 25 at f16 and 1/60 second. Should I work outdoors on a sunny day, would I not be able to eliminate the effect of ambient lighting by:

- using f22 with the flashes closer would then put the ambient light two stops lower than the flashes

- realize another stop diminishment from lens extension (already designed into the flash energy)

- shade the subject from direct sunlight

- perhaps use polarizing or ND filters with the flashes even closer (initially I have them being 1.5 ft. from the subject). This could put the ambient light four or more stops lower than that required for correct exposure.

Do you think an image due to that light would still register on the film? I don't have a good sense of the range of light values on slide film, from shadow to highlight.

I love the excerpt provided by Ed on National Geographic photographers. Makes me feel better about getting only 1 or 2 (or 0) satisfactory images on a roll!

Msg#:51611
From: JOHN MUCHOW To: MIKE RAPP

The focal plane shutters I've seen have the curtains run from bottom to top. When the flash duration of the strobes I'm using exceeds the full open time of the shutter (approx. 1.5 milliseconds at 1/250 sec. exposure], I get the image of the second curtain across the bottom of my shot. Of course, there are several brands of shutters and they might all run in different directions. I know...I know, nitpicking, we photographers are like that! :-}

Msg#:51612
From: JOHN MUCHOW To: DAVID MACDONALD

Using the lens extension and the ND or polarizing filter will reduce the light from the ambient source and your flash at the same time, giving you the same ratio between the sources, just at a lower light level. You have to get the flash as bright as possible (which you're doing by moving them closer and blocking direct sunlight), preferably at least 4 stops. Be sure your background is also not bright (a lot of nature guys use black velvet to help separate their subject, insect and flower for example, from the background clutter).

Kodachrome is quite a contrasty film. A ratio of more than 2 to 2-1/2 stops from highlight to shadow side of subject will begin to wash out highlights or blacken out your shadow detail. Ektachrome is a little more forgiving. Negative film, 'especially* Ektar 25, is very forgiving and the detail is incredible! It doesn't matter what film speed you use, just close down as far as possible, use the highest shutter speed that will sync to your flash, and reduce the ambient light level as much as possible and you'll be fine.

Msg#:51638
From: MIKE RAPP To: DAVID MACDONALD

Yes, anything you do to increase the ratio of controlled (strobe) light to ambient will help. Again, even outside, I would make every attempt to darken the background. At your close focus distances most of the background will be out of focus anyway. For a scientific study, a dark background is no problem. For "pretty" pictures you might consider some creative darkroom work to combine foreground subjects with separately shot backgrounds; it's more common than you may think!

Color slide film does not have as much range in light values as black & white, but Kodachrome 25 is about as good as you'll get in this regard.

One or two good shots per roll may not be that bad. One cost saving tip if you use commercial processing is to have the lab just develop the film and not do any mounting. You then just mount the few good shots. Nature photography is always a test of patience. Throw in close focus, rapid motion, and strobe sync and you have a severe challenge.

Msg#:51642
From: MIKE RAPP To: JOHN MUCHOW

In the old days (1970s) all 35mm focal plane shutters traveled from right to left with one exception. The oddball

CONNECTIME

was the Nikkormat, which was the "economical" Nikon of the day and used a vertical-moving bladed shutter that I think was called a Copal shutter if memory serves. The rest used a horizontally moving curtain of rubberized fabric except for the Nikon and Canon F1 which had titanium foil curtains. I know today's Nikon F4 uses a vertical shutter and achieves 1/8000 sec. shutter speeds. The rest of the industry may well have moved in that direction also. Perhaps some patents have expired. A vertical shutter would seem to have the advantage of only having to move 2/3 as far.

The partial image that you are seeing in flash shots faster than 1/250 is not because the flash duration is too long. Rather, it's caused by the fact that at speeds faster than 1/250 [or whatever a given camera's flash sync speed is) the closing curtain has started in motion before the opening curtain has reached full open. Since the signal is sent to the strobe at the instant the opening curtain reaches full open, you wind up with less than a full image. At maximum shutter speed you may see only 1/4 or less of the image.

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, or 2400 bps.

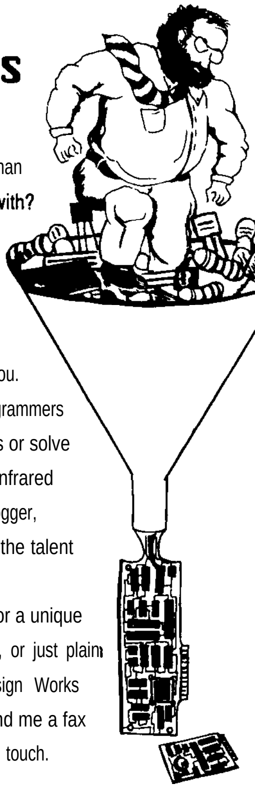
ARTICLE SOFTWARE

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

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

IRS

431 **Very Useful** **432 Moderately Useful** **433 Not Useful**



The Ciarcia Design Works

Does your big-company marketing department come up with more ideas than the engineering department can cope with? Are you a small company that can't afford a full-time engineering staff for once-in-awhile designs? Steve Ciarcia and the Ciarcia Design Works staff may have the solution for you. We have a team of accomplished programmers and engineers ready to design products or solve tricky engineering problems. Need an infrared remote controller, multi-channel data logger, or 7-axis drill controller? The team has the talent to design and manufacture it! Whether you need an on-line solution for a unique problem, a product for a startup venture, or just plain experienced consulting, the Ciarcia Design Works stands ready to work with you. Just send me a fax discussing your problem and we'll be in touch.

Remember...*a Ciarcia design works!*

Call (203)875-2199 • Fax (203) 872-2204

#221

OPTIMIZE YOUR MCU PROGRAM DEVELOPMENT !

INTEGRATE THE POWER **OF**

◆ **Editors** ◆ **Cross Assemblers** ◆ **Disassemblers** ◆
◆ **Cross Compilers** ◆ **Data Conversion Utilities** ◆
◆ **Simulators** ◆ **Serial Communications** ◆

ARMADILLO +™

A Unique, Universal Development / Communications Environment Supporting :

◆ All families of cross-assemblers and compilers.
◆ Communications with your target CPU.
◆ User definable utilities menu.
◆ Pull-down menus with mouse or keyboard control.
◆ IBM PC or compatible.

Now you can EDIT, ASSEMBLE, UPLOAD, DEBUG, and MORE, all from within ODE, FAST, ERSY-TO-USE MENU DRIVED ENVIRONMENT !

$99.00 + $2.00 P/H

TO ORDER CALL (804) 479-3893

LIFE FORCE TECHDOLOGY
5477 RUTLEDGE RD., VA. BEACH, VA. 23464

STEVE'S OWN INK

Making “Sense” of the World

It seems like the more I am able to achieve in implementing the Circuit Cellar Home Control System (HCS II), the more tasks seem to present themselves. I spent many years asserting the benefits of distributed control and here I am ripping out all the individual control systems I've installed in the last 8 years and replacing each of them with a few lines of HCS code.

The reality is that the HCS's vastly expanded I/O capacity and network capability allow it to make decisions based on a global connection to the process rather than just an isolated segment. A simple attic vent fan can be transformed into a useful air conditioning system when the HCS combines data on in/out differential temperatures, time of day, and door and window openings into a proportionally controlled temperature program. While sensing all these parameters directly is within the capability of a local controller, duplicating the process for more than a few independent positions is hard to justify. Sensing all parameters within one system and allocating the task to a networked controller is a more sensible goal.

Of course, setting a goal doesn't necessarily indicate the cost or effort in achieving it. Some control goals can be difficult. Let me explain.

I have a friend who is a police officer. As I was showing him the driveway video camera and recording system, he chuckled sheepishly.

‘I don't mean to diminish your achievement here, Steve, but with the recruits I'm seeing these days, you won't help arrest any perpetrators unless you have a picture of their license plate or of their face with their social security number tattooed across their forehead.”

Aha, a goal! Snapping a picture of a driver or license plate didn't appear to be a difficult task. After all, I had a powerful HCS to coordinate the action. All I needed was a few sensors and other stuff.

Unfortunately, taking a picture of a moving car is not as easy as it sounds. (Did I ever tell you about the jerk in the VW Bus who came down my driveway at 60 MPH [claimed he thought he was still on the highway], drove over a cliff, and ended up with his car hanging in the trees?) To further complicate matters, the camera not only had to work with minimal illumination at night but also had to produce pictures in the brightest sunlight or with headlights aimed at it without being destroyed at the same time. I finally concluded that capturing just the license plate would have to do. A fixed-iris, low-light-level CCD camera coupled with appropriate illumination at night (a 250-watt spotlight flashed on the back of the car) produced a decent picture.

OK, we have video, but have we accomplished the goal when its out in the garage 150' from the HCS? Where exactly do we present this snapshot? Well, to make a long story short, I did it both ways. Using a video digitizer from a previous Circuit Cellar project, I grabbed the camera output when the HCS sensed the interrupted IR beam and sent it serially back to the house as RS-422 (remember RS-232 is only good for 50 feet) where it was reconstructed as video and displayed on a monitor. At the same time I ran the camera's raw video, using the appropriate long line amplification and compensation electronics, to a monitor next to the other unit. Finally, to actually record the snapshot, I added a Sony hard copy video printer that could also be triggered by the HCS. Experimentation would determine whether the live video or digital frame grab would be the better picture.

In either case, the whole control code in the HCS was barely 20 lines to handle all the initialization, sensing, and printing.

When my friend showed up again and reviewed my achievement he was somewhat surprised by the efforts I went through to achieve it. Perhaps that really is a failure in the HCS. A system that is so easy to use promotes setting far-fetched goals while neglecting the potential cost of achieving them. Of course, since I have the excuse of all you Circuit *Cellar INK* readers when I do something, I can call it entertainment. My friend had a few alternative choice descriptions.

Steve