A CAHNERS PUBLICATION

SEPTEMBER 17, 1990

EDN®

ELECTRONIC TECHNOLOGY FOR ENGINEERS AND ENGINEERING MANAGERS

SOFTWARE ENGINEERING SPECIAL ISSUE

SUPPLEMENT TWO EDN

At last, an entirely new approach to clock speed

Get five times faster throughput from NEC K-Series™ microcomputers.

As a developer of real-time control systems, you know that designing in a faster CPU is not enough. You also need intelligent I/O management for the best possible system throughput.

NEC's K-Series™ microcomputers are perfect for real-time control designs requiring multitasking, such as automotive control, ISDN and computer peripheral controllers.

Peripheral Management Unit™

The K-Series' unique architecture includes a revolutionary Peripheral Management Unit™ macro service for nonstop instruction execution while processing up to 16 I/O requests at the same time. By designing in the K-Series microcomputer, you can improve your system throughput by as much as 5X.

The K-Series 8-bit and 16-bit microcomputers give you a real-time output port; an advanced counter/timer system; a high-speed, high-resolution A/D converter; and many other on-chip intelligent peripherals.

Not since the invention of the hourglass has anyone come up with a more ingenious way to speed up silicon.

The K-Series provides you a worry-free upgrade path from the 8-bit K2 microcontroller family to the 16-bit K3 devices. And your future designs will exploit the power of the lightning-fast 125-ns K6, with real-time operating system in microcode, and complete K3 software compatibility.

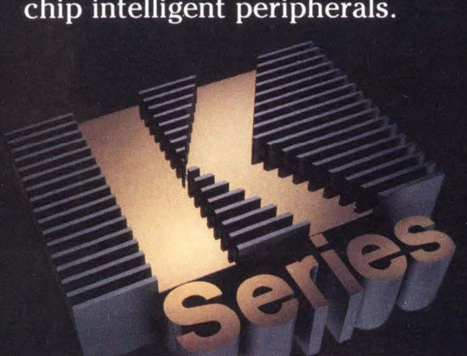
Free Knowledge Kit™

To learn more about the K-Series microcomputers with up to 1K bytes of on-board RAM, 32K bytes of ROM/EPROM, and Peripheral Management Unit coprocessing power, call now.

1-800-632-3531

A free 81-page Knowledge Kit™ will be rushed to you along with complete information on the K-Series with the Peripheral Management Unit. Development tools and samples are available now.

NEC Electronics Inc.
401 Ellis Street
Mountain View, CA 94039-7201
1-800-632-3531/415-965-6158



Embedded Control

NEC

CIRCLE NO. 22

All the features of HPBASIC, and more.

For less.

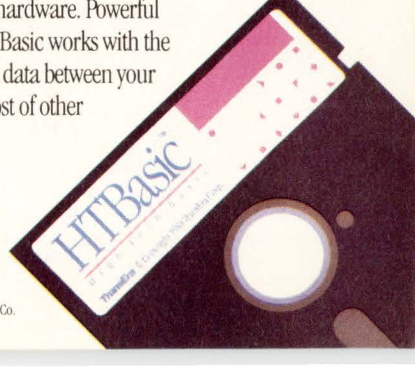
HTBasic	BASIC FEATURES:	HP BASIC
YES	IEEE-488 GPIB (HP-IB), RS-232 Instrument Control	YES
YES	Integrated Environment: Mouse, Editor, Debugger, Calculator	YES
YES	Supports 16 Megabytes of Memory (breaks DOS 640K barrier)	YES
YES	Engineering Math: Matrix Math, Complex Numbers	YES
YES	High Level Graphics: Screen, Plotter, Printer	YES
YES	Structured Programming with Independent Subprograms	YES
YES	Runs on Industry Standard Personal Computers	NO*
YES	Industry Standard Graphic Printer Support: Epson, IBM, lasers, etc.	NO
YES	Industry Standard Network Support: Novell, IBM, Microsoft, NFS, etc.	NO
YES	Industry Standard IEEE-488 Support: National Instruments, IOtech, etc.	NO
YES	Exchange data files with Industry Standard PC applications	NO*
YES	No-charge Telephone Technical Support	NO
YES	Instant on-line HELP system	NO

A Costly Situation. Every engineer needs the power and features of a "Rocky Mountain" BASIC workstation, but not everyone can have one. They simply cost too much. Fewer workstations, less productivity. **The Best Way.** TransEra HTBasic software provides the *only* way for serious technical computer users to turn their PC into a workstation without having to add costly hardware. Powerful workstations for everyone means greater productivity. **Extraordinary Versatility.** In addition, TransEra HTBasic works with the Industry Standard Personal Computer hardware, software, and networks. It even allows you to easily exchange data between your favorite DOS programs and the files you create in the BASIC workstation environment. All at a fraction of the cost of other solutions.

More compatibility. More versatility. More possibilities. Less expense. Less hassle.

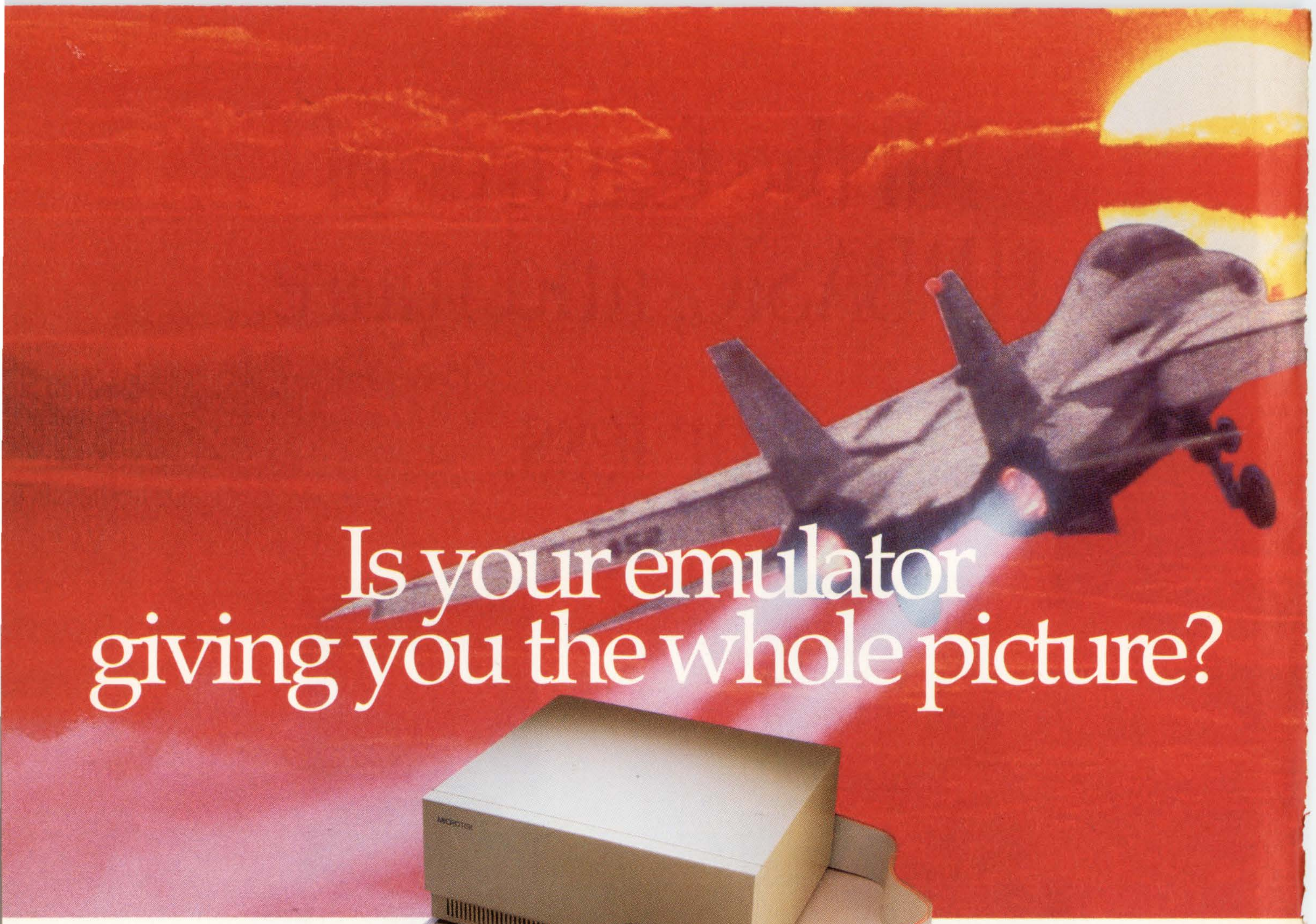
To find out more, call 1-801-224-6550.

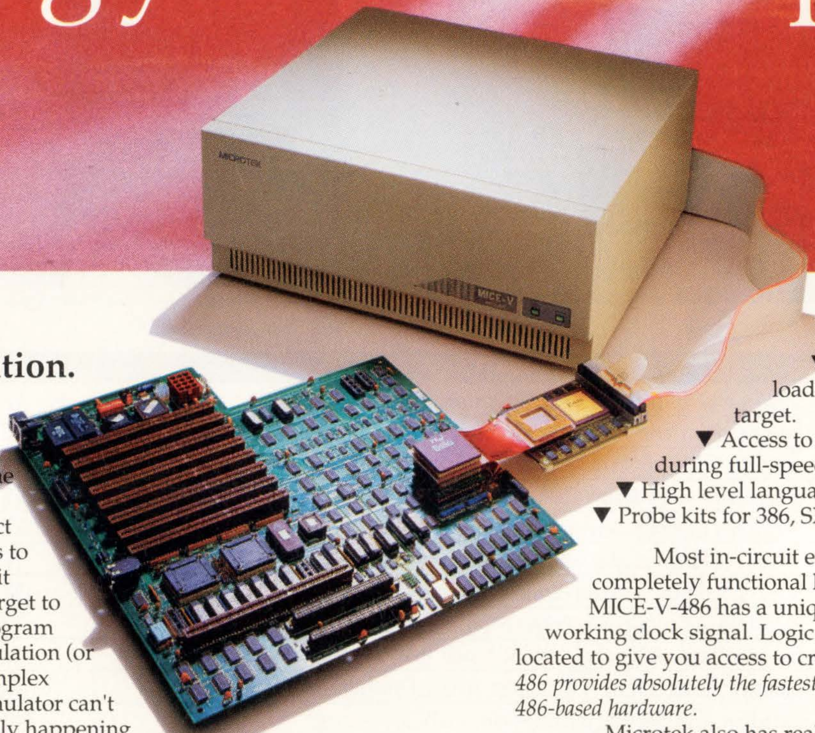
TransEra
Engineering Excellence for 15 Years ™



CIRCLE NO. 1

* Without the addition of a costly 68000 co-processor.





MICE-V-486.
33MHz Emulation.
Real features.
Real-time.

Without real-time emulation you never know how your product will perform until it has to fly. Traditional in-circuit emulators slow your target to collect, display or reprogram trace. Or even stop emulation (or your target) to load complex triggers. When your emulator can't show you what's actually happening you risk missing a bug that will sneak from your prototype to the finished product.

MICE-V-486 lets you see it all.

- ▼ Real-time emulation to 33MHz.
- ▼ Complex, sequential triggers, loaded without slowing the emulator or target.
- ▼ Access to the fully qualified trace buffer during full-speed emulation.
- ▼ High level language debug.
- ▼ Probe kits for 386, SX, 376 and 286 support.

Most in-circuit emulators require partially or completely functional hardware to operate correctly. MICE-V-486 has a unique *Isolation Mode*™, requiring only a working clock signal. Logic analyzer taps are conveniently located to give you access to critical timing information. *MICE-V-486 provides absolutely the fastest method for debugging non-functional 486-based hardware.*

Microtek also has real-time emulators and source-level debuggers for 68000, -020, -030 and 80C186.

So, stop wasting development time because your emulator isn't real-time. Call us, and get your product to market fast.

MICROTEK

The Leader In Development Systems Technology.™

MICROTEK INTERNATIONAL, INC. - Development Systems Division
3300 N. W. 211th Terrace, Hillsboro, OR 97124 • (503) 645-7333 • Fax (503) 629-8460

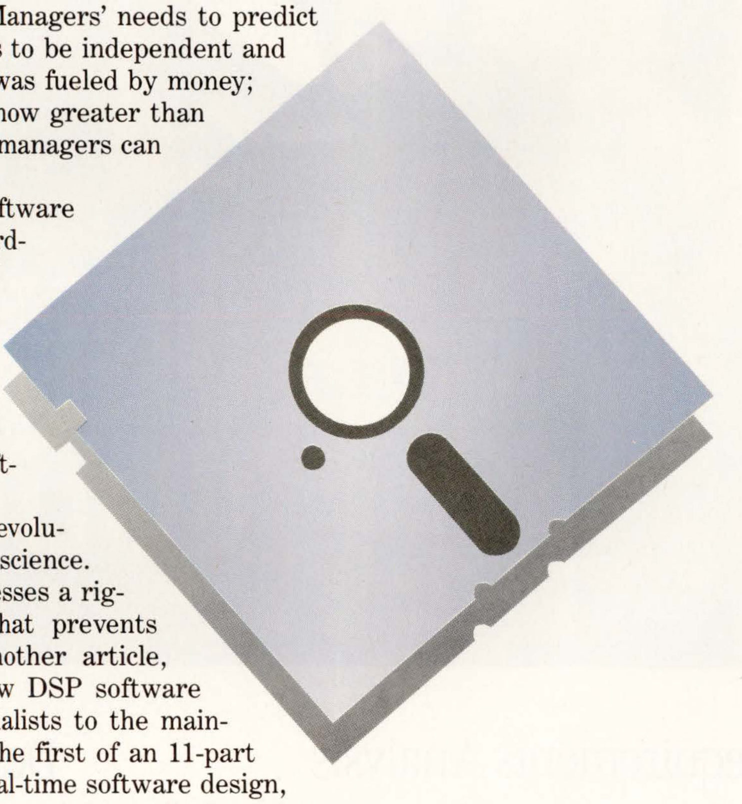
ASIA OFFICE – Taiwan - MICROTEK INTERNATIONAL 886-2-723-5577/Japan - CORE Digital 81-3-7955171
EUROPE OFFICE – Germany - ALLMOS 49-89-8570000/France - M.T.E. 33-1-39618228/U.K. - ARS Microsystems 44-276-685006

CIRCLE NO. 2

Art becomes science

Not so very long ago, software development was more an art than a science. The best programmers were those brilliant individuals who could disappear into small cubicles, work their creative magic, and then reappear some weeks or months later with programs of dazzling cleverness. Often, though, the programs were so clever no one could understand them. Sometimes they did what they were designed to do, but were designed to do the wrong thing. The fact that a program did the wrong thing with dazzling cleverness was usually of some satisfaction to the programmer, but very little to the programmer's manager.

So it happened that software development—mostly an art—led to software engineering—mostly a science. Managers' needs to predict results won out over programmers' desires to be independent and creative. As with many changes, this one was fueled by money; the value of software in most products is now greater than the value of the hardware, a fact that few managers can afford to overlook.

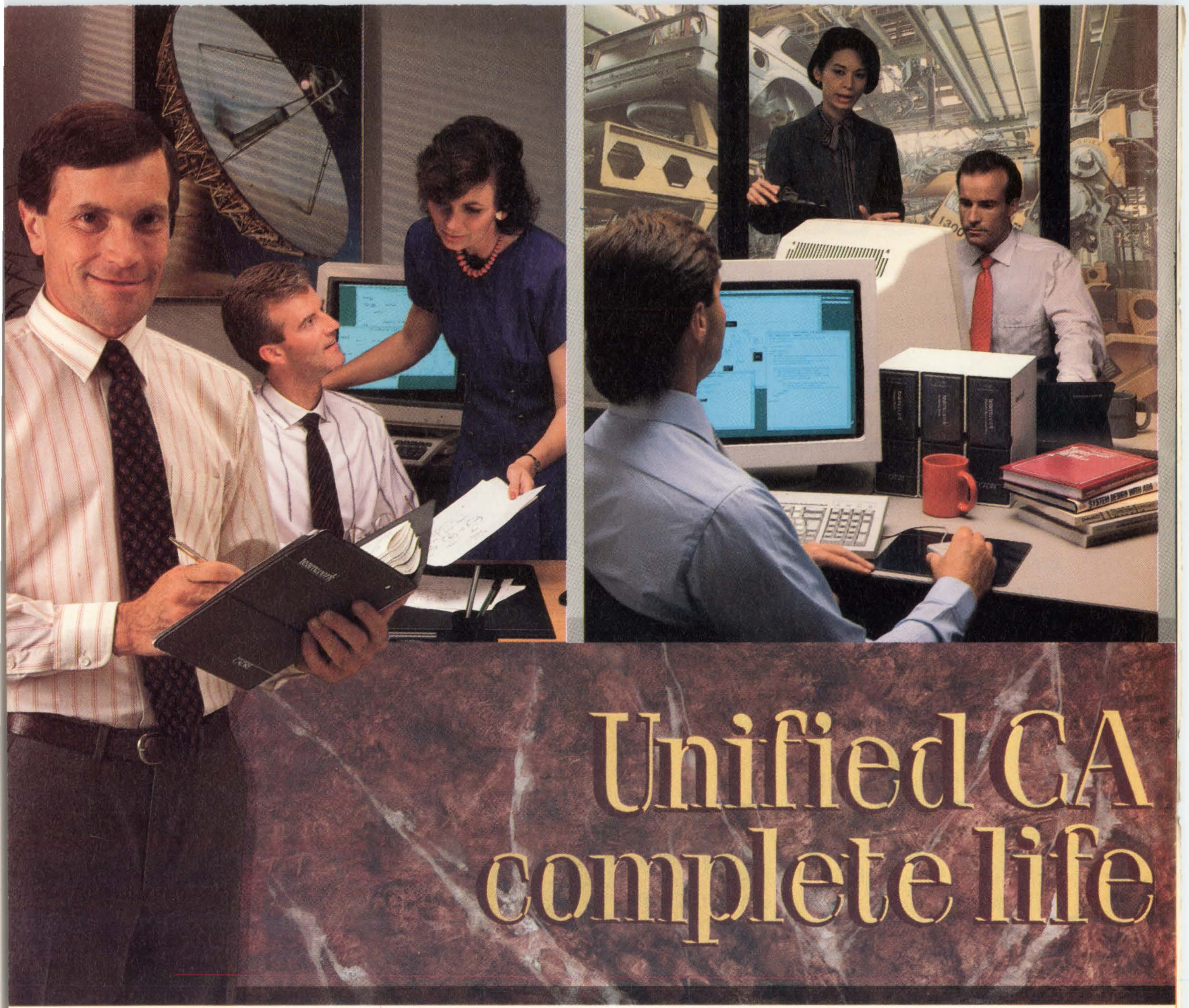


Software's growing importance and software engineering's increasing overlap with hardware engineering have led the editors of EDN, over the years, to increase their software coverage. In this special issue, we give you nothing but software. It's not software in a vacuum, though; rather, it's the kind of software that also involves hardware, the kind of software EDN readers are involved with.

The articles included here all reflect the evolution of software development from art to science. An article on debugging, for example, stresses a rigorous approach to designing software that prevents many problems from ever occurring. Another article, on DSP software development, shows how DSP software is moving from an esoteric group of specialists to the mainstream of software development. And in the first of an 11-part series, we begin a no-nonsense guide to real-time software design, previously one of the blackest black magics of them all. Rounding out the list of technical articles are a practical guide to developing OS/2 applications and some useful techniques for combining C and assembly language.

You could view these articles as a few small components in the enormous structure that is software-as-science. In science, you have a knowledge base to build on; you don't have to waste creative energy reestablishing what others have already done. But creativity is as essential to science as it is to art, and the rules of science should help us, not hinder us; if they don't, we should break them. Rules were made to be broken, after all, and even science should be fun. If programmers lose the spirit that existed when software was more an avocation than a vocation, when it was more an art than a science, then we will all feel the loss.

Gary Legg
Special Projects Editor



Requirements Analysis

Cadre's Unified CASE product family brings software design automation to the entire systems development lifecycle, from requirements analysis through to product test. Requirements analysis extracts product function and behavior from fundamental system requirements. Workstation-based Teamwork and ADAS describe system requirements and evaluate tradeoffs between cost and performance.

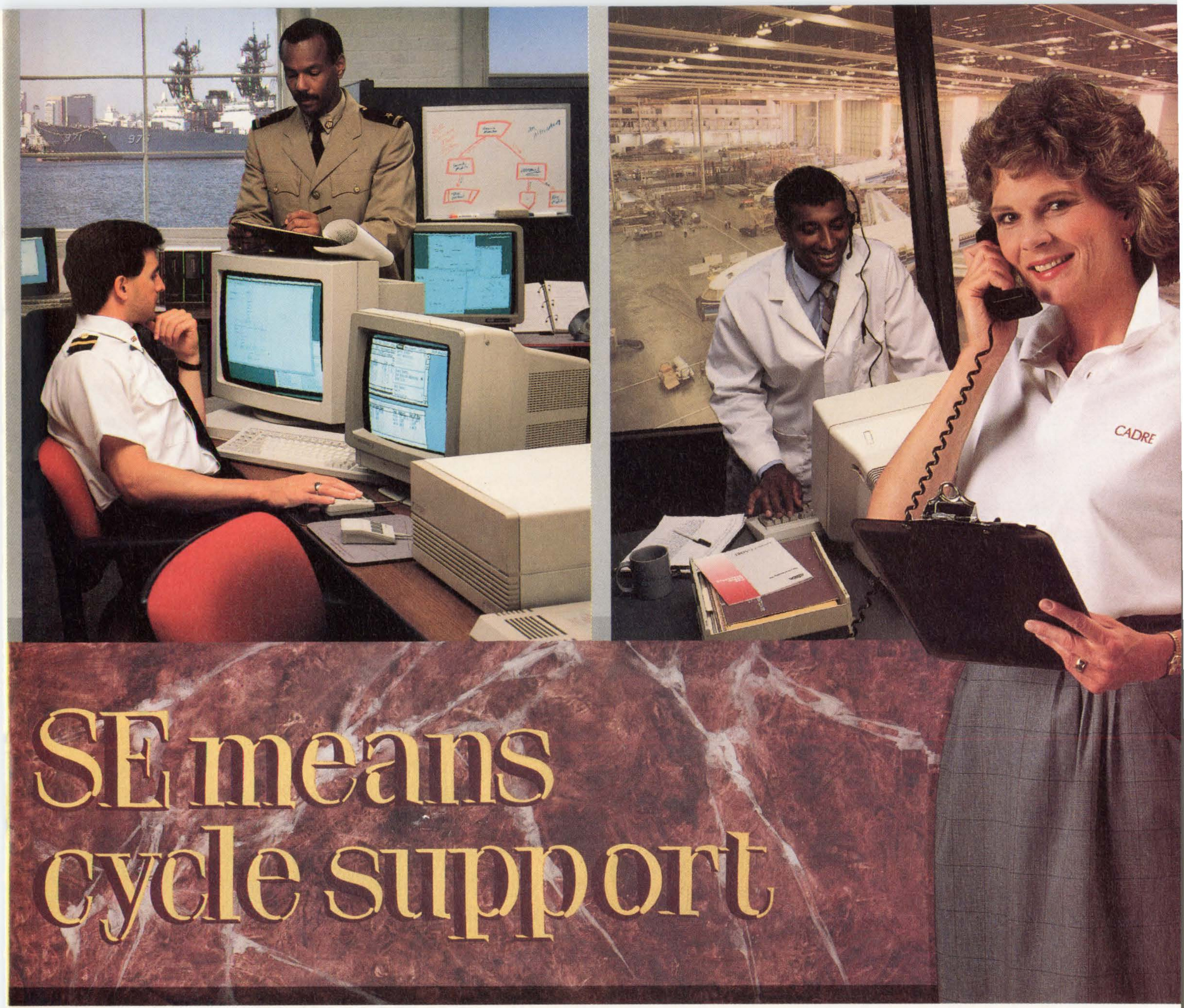
Design & Code

With requirements analysis completed, project teams implement software design and program logic using Teamwork. Design converts directly to code using Ada and C Source Builders, improving consistency and product maintainability. Cadre's networked Unified CASE environment moves design teams quickly through the product lifecycle to develop superior products.

teamwork®
Winning teams depend on it.

CADRE means

PathMap, PROBE, Software Analysis Workstation, Unified CASE, Ada Source Builder, and C Source Builder are trademarks of Cadre Tecl



SE means cycle support

Test & Integration

Cadre's Unified CASE test products debug and test embedded software directly in target hardware. Test data automatically annotates original design for complete traceability. Cadre's Software Analysis Workstation, or SAW, measures execution and performance of embedded software. PathMap, Cadre's Run-time Reverse Engineering tool, creates design-level Structure Charts from actual code executing in target hardware. PROBE in-circuit emulators allow software engineers to operate from their own workstation platforms.

Ongoing Product Support

The Unified CASE product family includes Cadre's responsive customer service, specialized training programs, consulting services, and ongoing product enhancements. How can Unified CASE improve your products? To find out

Call (401) 351-CASE

When critical systems need to be developed, project leaders call for Unified CASE solutions. Available only from Cadre.

CIRCLE NO. 3

Unified CASE

CADRE

Cadre Technologies Inc.
222 Richmond Street, Providence, RI
02903 U.S.A. (401) 351-2273

Winning teams depend on us.

Emulation power without compromise

EZ-Pro™ 1.5 price performance leader for 8-bit in-circuit emulation.

EZ-Pro 2.1 industry workhorse for 16-bit and 8-bit designs.

Power in selection—System support for more processors than any other manufacturer in the world. Power in product range to match your needs—from economical basic configurations to fully featured systems.

Power in performance—Completely integrated capabilities include options such as versatile trace, performance analysis, EPROM programming, C source level debugging, over 100 personality modules with a common universal platform for different processors, C cross compilers, cross assemblers and more.

Power without compromise—All invented here. Supported here. And available to rent or purchase now.

Free Demo Disk!

See how easily you can use these sophisticated development tools. Our marketing department will ship your demo disk today. Please Call:

(714) 731-1661

american automation **Headquarters:** 2651 Dow Avenue, Tustin, California 92680-7207 Telephone (714) 731-1661. **European Headquarters:** UK Oxford 993 778991. **Distributors:** Australia 3-5601011, Belgium 2-4681400, France 1-69308050, India 418387, Indonesia 22-71880, Italy 2-50722282, Korea 2-7849942, Spain 1-7291155, Switzerland 1-4354111, Taiwan 2-7368150, West Germany 89-6127087.

EZ-Pro is a trademark of American Automation Inc.

CIRCLE NO. 4

Volume 35, Number 19B

EDN® Magazine Edition

September 17, 1990

ELECTRONIC TECHNOLOGY FOR ENGINEERS AND ENGINEERING MANAGERS

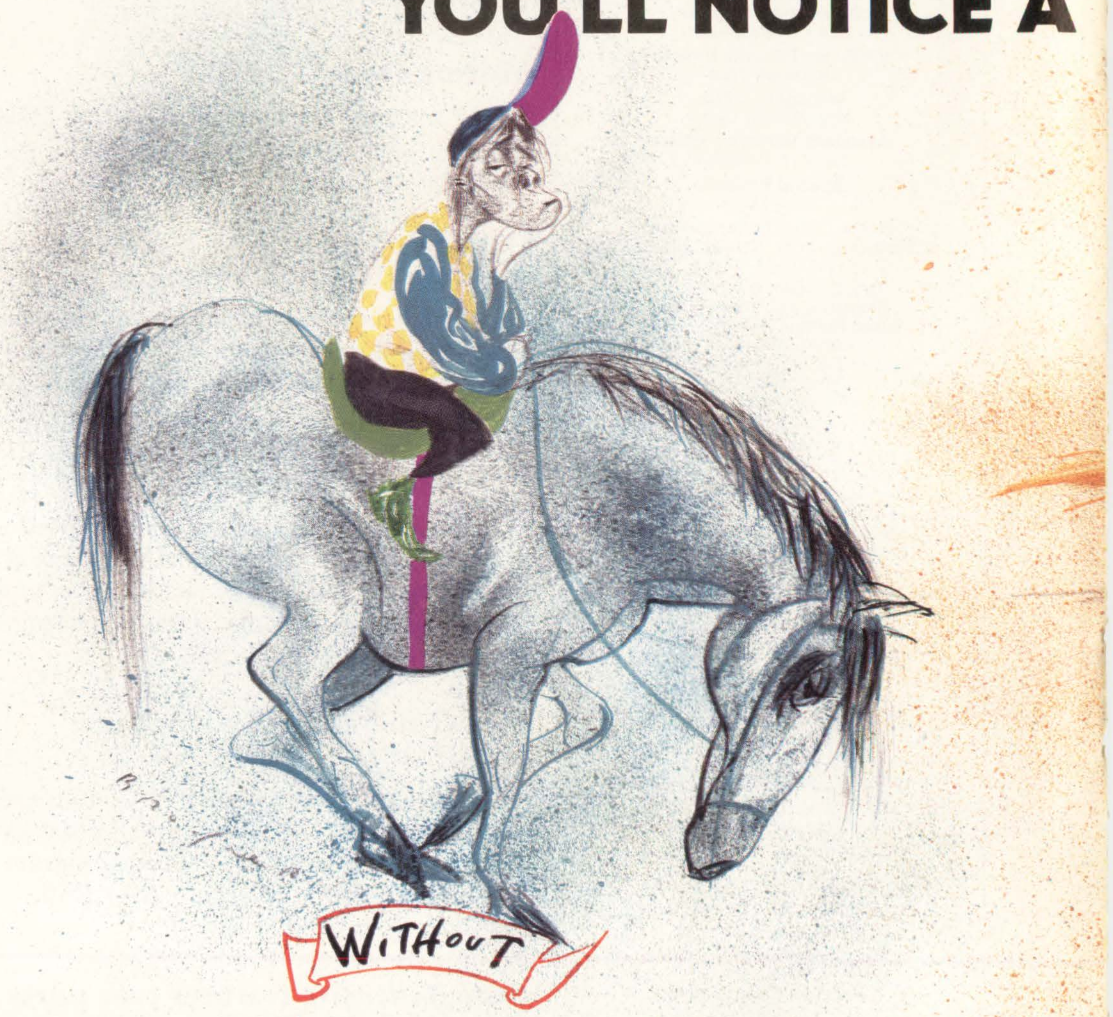
VP/Publisher
Peter D Coley
Associate Publisher
Mark J Holdreith
VP/Editor/Editorial Director
Jonathan Titus
Managing Editor
Joan Morrow Lynch
Assistant Managing Editor
Susan L Rastellini
Special Projects
Gary Legg
Home Office Editorial Staff
275 Washington St, Newton, MA 02158
(617) 964-3030
Tom Ormond, *Senior Editor*
Charles Small, *Senior Editor*
Susan Bureau, *Associate Editor*
Jay Fraser, *Associate Editor*
John A Gallant, *Associate Editor*
Michael C Markowitz, *Associate Editor*
Dave Pryce, *Associate Editor*
James P Scanlan, *Associate Editor*
Julie Anne Schofield, *Associate Editor*
Dan Strassberg, *Associate Editor*
Chris Terry, *Associate Editor*
Helen McElwee, *Senior Copy Editor*
Christine McElvenny, *Senior Production Editor*
Gabriella A Fodor, *Production Editor*
Brian J Tobey, *Production Editor*
Editorial Field Offices
Steven H Leibson, *Senior Regional Editor*
Boulder, CO: (303) 494-2233
Doug Conner, *Regional Editor*
Atascadero, CA: (805) 461-9669
J D Mosley, *Regional Editor*
Arlington, TX: (817) 465-4961
Richard A Quinnell, *Regional Editor*
Aptos, CA: (408) 685-8028
Anne Watson Swager, *Regional Editor*
Wynnewood, PA: (215) 645-0544
Maury Wright, *Regional Editor*
San Diego, CA: (619) 748-6785
Brian Kerridge, *European Editor*
(603) 630782
(St Francis House, Queens Rd,
Norwich, NR1 3PN, UK)
Contributing Editors
Robert Pease, Don Powers,
David Shear, Bill Travis
Editorial Coordinator
Kathy Leonard
Editorial Services
Helen Benedict
Art Staff
Ken Racicot, *Senior Art Director*
Chinsoo Chung, *Associate Art Director*
Cathy Madigan, *Staff Artist*
Production/Manufacturing Staff
Andrew A Jantz, *Production Supervisor*
Sandy Wucinich, *Production Manager*
Deborah Hodin, *Production Assistant*
Kelly Brashears, *Production Assistant*
Diane Malone, *Composition*
Director of Art Department
Joan Kelly
Norman Graf, *Associate*
VP/Production/Manufacturing
Wayne Hulitzky
Director of Production/Manufacturing
John R Sanders
Business Director
Deborah Virtue
Marketing Communications
Anne Foley, *Promotion Manager*
Pam Winch, *Promotion Assistant*

SOFTWARE ENGINEERING SPECIAL ISSUE

EDN®(ISSN 0012-7515) is published 50 times a year (biweekly with 2 additional issues a month, except for February and September, which have 3 additional issues and July and December which have 1 additional issue) by Cahners Publishing Company, A Division of Reed Publishing USA, 275 Washington Street, Newton, MA 02158-1630. Terrence M McDermott, President; Frank Sibley, Senior Vice President/General Manager, Boston Division; Jerry D Neth, Senior Vice President/Publishing Operations; J J Walsh, Senior Vice President/Finance; Thomas J Dellamaria, Senior Vice President/Production and Manufacturing; Ralph Knupp, Vice President/Human Resources. Circulation records are maintained at Cahners Publishing Company, 44 Cook Street, Denver, CO 80206-5800. Telephone: (303) 388-4511. Second-class postage paid at Denver, CO 80206-5800 and additional mailing offices. POSTMASTER: Send address corrections to EDN®, PO Box 173377, Denver, CO 80217-3377. EDN® copyright 1990 by Reed Publishing USA; Ronald G Segel, Chairman and Chief Executive Officer; Robert L Krakoff, President and Chief Operating Officer; William M Platt, Senior Vice President. Annual subscription rates for nonqualified people: USA, $109.95/year; Canada/Mexico, $135/year; Europe air mail, $165/year; all other nations, $165/year for surface mail and $250/year for air mail. Single copies are available for $10. Please address all subscription mail to Ellen Porter, 44 Cook Street, Denver, CO 80206-5800.

SAMSUNG NOW

68040 SYSTEMS EQUIPPED YOU'LL NOTICE A



Systems utilizing the new Samsung 84C31 take off. They run like Triple Crown-winning thoroughbreds. They blaze, scorch, and leave others in their dust.

In a word, they are fast.

And they make even speedy 68040 systems that *don't* use the 84C31, look like they're not in motion.

The 84C31 was designed with the close cooperation of Motorola. It is the *only* DRAM controller designed specifically for Motorola's powerful 68040 and 68030 microprocessors.

SAMSUNG'S DRAM CONTROLLER FOR THE 68040 AND 68030

Part Number	*RAMs Supported*	*Package*
KS84C31-33CL	256K, 1Mb	68-pin PLCC
KS84C32-33CL	256K, 1Mb, 4Mb	84-pin PLCC

Samples and production available now.

Like the extremely successful earlier-generation Samsung System Accelerators™, the part is highly integrated and inherently fast. And as the cutting edge in memory control, it can help you simply and economically enhance even 68040 performance.

The 84C31 supports both the burst and non-burst modes of the 68040. It also provides a direct interface

WITH OUR DRAM CONTROLLER: CERTAIN IMPROVEMENT.

to the microprocessor. Which saves you dollars, board real estate, and

68030 PERFORMANCE SUMMARY

Access Clocks	*DRAM Speed*	*Frequency (Mhz)*
4-2-2-2	70 ns	20
5-2-2-2	120 ns	20
5-2-2-2	80 ns	25
6-2-2-2	120 ns	25
6-2-2-2	80 ns	33
7-2-2-2	100 ns	33

68040 PERFORMANCE SUMMARY

Access Clocks	*DRAM Speed*	*Frequency (Mhz)*
3-2-2-2	80 ns	25
5-2-2-2	100 ns	25
6-2-2-2	120 ns	25
5-2-2-2	80 ns	33
6-2-2-2	100 ns	33

design time, since it means you don't need additional glue logic.

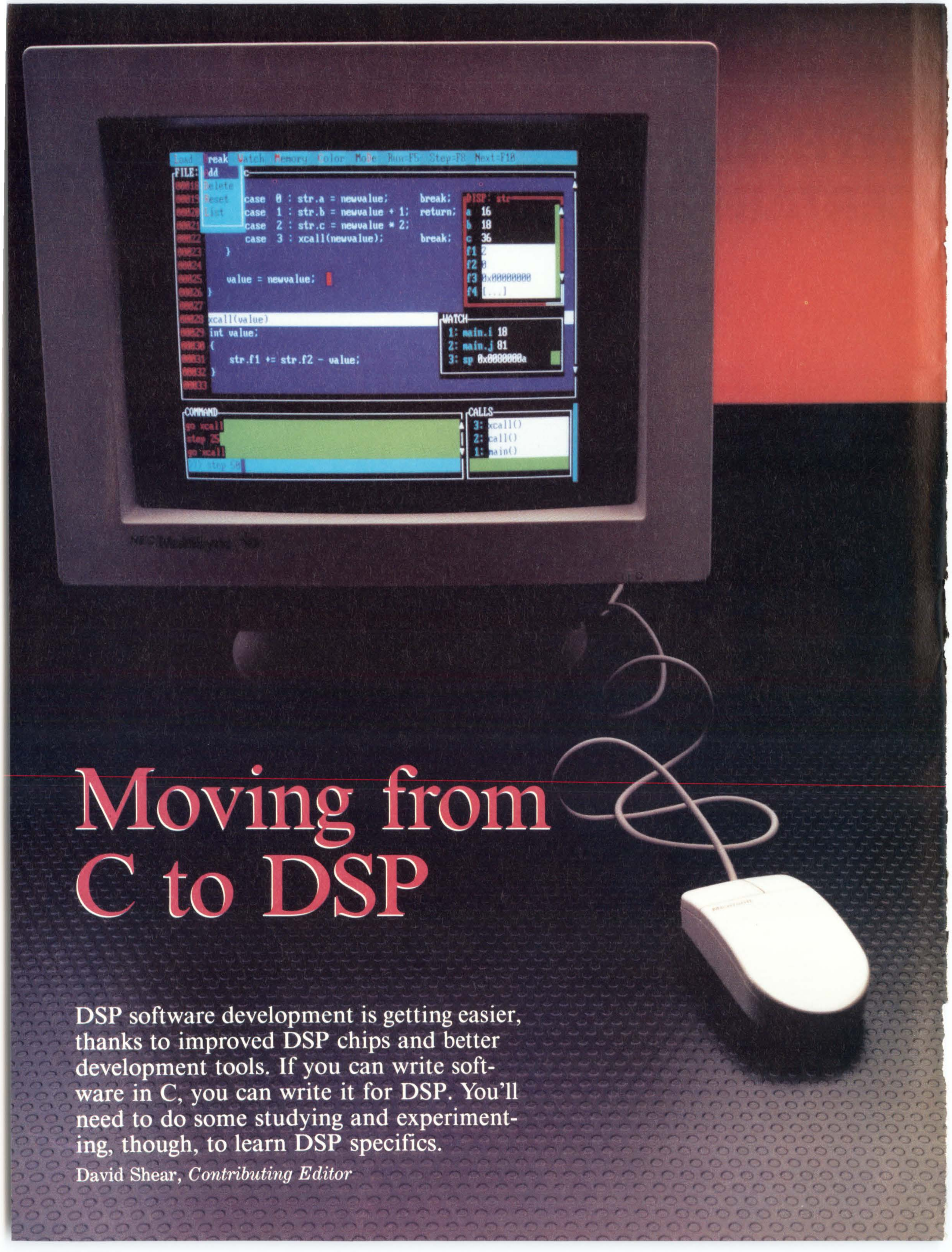
Ease of design is another advantage. As a glance at our System Design Guides will show, it's an unusually simple chip to design in.

All in all, we believe the 84C31 is the best memory controller solution available today.

For details on using it to make *your* designs take off, contact DRAM Controller Marketing, Samsung Semiconductor, 3725 No. First St., San Jose, CA 95134. Or call 1-800-669-5400, or 408-954-7229.

SAMSUNG
Semiconductor

CIRCLE NO. 5



```
        case  0 : str.a = newvalue;       break;
        case  1 : str.b = newvalue + 1;   return;
        case  2 : str.c = newvalue * 2;
        case  3 : xcall(newvalue);        break;
    }

    value = newvalue;
}

xcall(value)
int value;
{
    str.f1 += str.f2 - value;
}
```

```
a  16
b  18
c  36
f1 2
f2 0
f3 0x00000000
f4 [...]
```

```
1: main.i 18
2: main.j 81
3: sp 0x0000000a
```

```
go xcall
step 25
go xcall
step 50
```

```
3: xcall()
2: call()
1: main()
```

Moving from C to DSP

DSP software development is getting easier, thanks to improved DSP chips and better development tools. If you can write software in C, you can write it for DSP. You'll need to do some studying and experimenting, though, to learn DSP specifics.

David Shear, *Contributing Editor*

Digital signal processing (DSP) is moving into the mainstream. Low-cost DSP chips are going into products ranging from games to cellular telephones and from modems to electronic test equipment. When—not if—you face a DSP software project, you must know how to tackle it. Fortunately, new chips and their supporting development tools are making that task easier than it was a few years ago. For experienced C programmers, DSP projects require only a few new skills.

Advice for getting started in DSP ranges from "hire an expert" to "just open a book and have at it." Warren Cope, senior applications engineer at Spectrum Signal Processing, suggests starting with literature, routines, and application libraries from manufacturers and the public domain. "Call all the chip vendors and get whatever application literature they have," he says. "Just because you're using a TI chip doesn't mean you can't read Motorola's books. The algorithms still work no matter what chip you're using."

Ray Simar, principal architect and program manager for Texas Instruments' TMS320C30 and TMS320C40, echoes Cope's advice. For information on any of TI's DSP products, he suggests calling the Texas Instruments hot line ((713) 274-2320). In addition to customer-support phone lines, major chip vendors have computer-accessible bulletin boards with DSP information and program listings. Many textbooks also include DSP algorithms and listings in Fortran or C. Whatever method you use to acquire it, a general understanding of DSP is essential for developing DSP software (see **box,** "A DSP definition").

With some knowledge of DSP in hand, you'll want to learn about development tools for different DSP chips. The availability and quality of development tools should influence your decision about which DSP chip to use, because most DSP projects are software intensive. Some chips have few or no tools, and others have an impressive variety. Texas Instruments leads the industry in tools and assistance; Analog Devices, AT&T, and Motorola also offer a number of tools. Look closely at the tools available for a particular DSP chip before selecting it; the effort you waste when you don't have good tools can be enormous.

Because first-generation DSP chips had novel architectures, development tools for them were difficult to create and, therefore,

C source-level debuggers, like the TMS320C30 C Source Debugger from Texas Instruments (left), are changing the way you debug DSP software. You no longer have to look at an assembly listing to figure out what the compiler has done with your high-level code.

New hardware and software tools are changing the way DSP programs are developed.

scarce. These early chips had architectures optimized for DSP, and little else; capabilities for running the general-purpose part of a program were very limited. Modern DSP chips, however, are capable general-purpose processors, and the hardware and software tools for them are changing the way DSP programs are developed. Now, developing a DSP program resembles the development of other real-time programs.

The tools for a particular DSP chip used to include only an assembler and a simulator. Today, some DSP chips have an optimizing C or Ada compiler, a source-level debugger, a real-time emulator, a block-level-language programming system, a variety of code generators, and extensive DSP function libraries. Several vendors offer complete, dedicated DSP development systems that include all these tools. There's even a real-time multitasking operating system especially for DSP chips—Spox from Spectron Microsystems.

C vs assembly code

In addition to development tools, you must also consider the language in which you'll write your DSP programs. The advantages of C over assembly language are the same for a DSP chip as for a general-purpose μP: ease of programming, code readability, and reduced maintenance costs. Likewise, C has the same

A DSP definition

The definition of digital signal processing (DSP) is changing. Originally, DSP was defined as the processing of signals by digital means, as opposed to analog means. A signal was just that: a received sonar or radar pulse, an acoustic signal, a signal on aphone line.

Now a signal can be just about anything. It is simply a continuous stream of information. A common signal is the music that travels through your stereo. Data about the price of a stock over some period of time can be a signal. Any block of related data can be a signal.

Processing a signal means modifying the signal so you can understand the information it contains. In a stereo, the bass and treble controls process the signal to accurately reproduce the music. A sonar system processes a received signal to identify the location of targets. You can analyze the performance of a stock to identify trends. A modem processes transmitted and received signals so computers can communicate over phone lines. Any time a signal gets modified, signal processing occurs.

Signals can be processed by analog or digital techniques. Digital techniques began to develop when affordable digital computers became available in the 1960s. But the number and size of computers required for the digital approach limited its use to research and very specialized applications. The cost of using DSP has declined, however, and the number of available algorithms has increased to a point where you may soon be called upon to use DSP.

The signals you process with DSP don't have to be analog. For example, DSP can be helpful any time you need to crunch numbers. By arranging your data in the most beneficial way and using an appropriate algorithm, you can often manipulate your data very quickly.

Algorithms are problem-solving procedures that generally involve repetition of an operation. The FFT, or fast Fourier transform, is a common DSP algorithm that converts data from the time domain to the frequency domain or vice versa. A signal that looks very complex in the time domain may look simple in the frequency domain. If you use an oscilloscope to view a signal representing the noise from a machine, the scope trace probably won't make much sense. In the frequency domain, however, the signal may appear as only a few separate frequencies mixed together. The FFT conversion from the time domain to the frequency domain makes the data easier to understand.

DSP algorithms often change a large amount of complex data into a form that is easier to deal with. A vision system, for example, examines and identifies objects—perhaps good and defective products on a conveyer belt—and may reduce an incredible amount of data to a simple pass/fail result.

The algorithms you need for a project may be given to you, or you may have to dig into DSP literature to find them. You shouldn't have any trouble finding algorithms, though; many people make careers out of creating new algorithms and publishing them.

Listing 1—DSP program written in C

```
#include <stdio.h>
#include <libap.h>
#define N 1024              /* Number of points in the window */
#define M 10                /* log2(N), N = 2^M */

main ()
{
     static float freq_array[N/2];

     /* The other part of your program goes here. When you need
          to use the spectrum analyzer, use the following
          function call. */

     get_spectrum (freq_array);

     /* Use the frequency domain data acquired by get_spectrum */
}

/* get_spectrum  -  spectrum analyzer
                 This routine will read the input from an ADC,
                 perform a Hanning window on the data, do an
                 FFT, and convert the resulting complex data
                 into floating point amplitude.  Data returned
                 is an array of amplitude of type float in the
                 frequency domain.  The function read_adc will
                 cause the hardware controlling the ADC to gather
                 data at precise time intervals.
*/

get_spectrum (freq_array)
{
     register float *data, real, imaginary, ampsqrd, *freq;
     register int i;
     static float data_array[2*N];

     data = data_array;
     for (i = 0; i < N; i++)           /* read buffer full of data */
     {
          *data++ = read_adc();
          *data++ = 0.0;               /* all complex values = 0 */
     }
     chann0 (N,M,data_array);          /* complex Hanning window */
     fft (N,M,data_array);             /* FFT of the data */
     data = data_array;
     freq = freq_array;
     for (i = 0; i < N/2; i++)         /* get the amplitude */
     {
          real = *data++;
          imaginary = *data++;
          ampsqrd = (real * real) + (imaginary * imaginary);
          *freq++ = sqrt(ampsqrd);
     }
}
```

disadvantages: increased execution time and program size. You'll probably want to use C as much as possible and assembly language only for those sections that require greater speed.

Depending on your application, you may be able to program an entire project in C. **Listing 1**, for example, shows a simple DSP program that performs a spectrum-analyzer function. The program makes function calls to an application library of optimized assembly-language routines.

Unfortunately, it is difficult to determine how much a high-level language will degrade your system's performance. Optimizing compilers—available from Analog Devices, AT&T, Intermetrics, Motorola, and Texas Instruments—are an improvement over standard C compilers, but they still don't produce code that's as efficient as that written by an experienced assembly-language programmer.

To begin DSP programming, you don't even need a DSP chip. You can compile and run algorithms on your PC. They won't run as fast on a PC's general-purpose μP, but they'll give you some useful insight. After experimenting with DSP algorithms on a PC, you can increase capability by adding a plug-in board that has a DSP chip. Many companies make such boards. Texas Instruments, for example, sells the $995 EVM, a PC card with a TMS320C30 DSP chip, memory, an ADC, and a DAC. The EVM works with all of TI's software development tools. An assembler/linker and source-level debugger come with the card; a C compiler is optional.

You can also gain DSP experience by using a DSP function library written in C, such as DSPL from Sonitech International. Its algorithms won't run as fast as assembly-language versions, but they're easier to understand. If the routines aren't fast enough for your final product, you can still use them for prototyping and then optimize some of the code as you become more familiar with the DSP chip you're using.

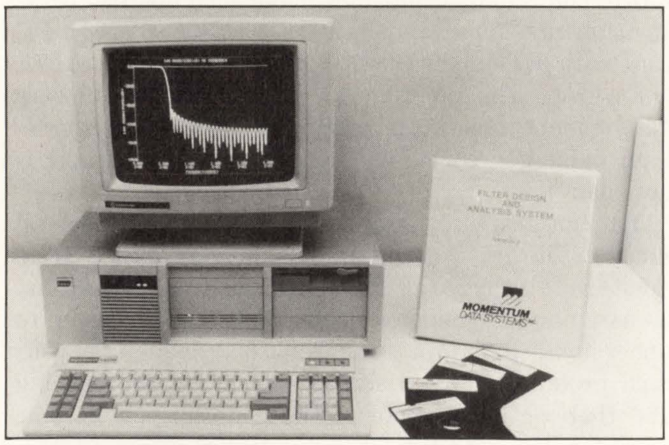
Another helpful tool for learning about the signal-processing aspect of DSP is filter-design software. Many available packages will accept your specification

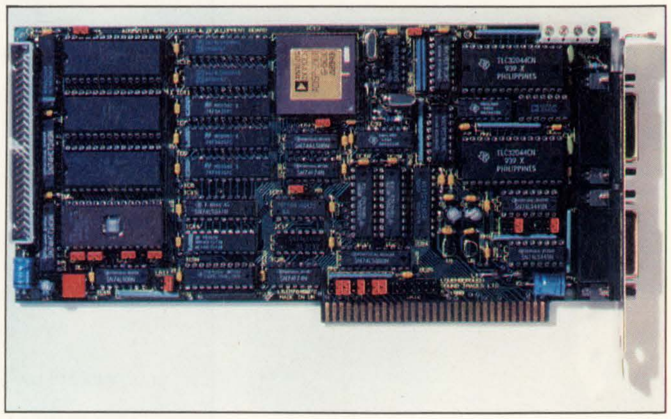
Depending on your application, you may be able to program an entire DSP project in C.

for a filter and then calculate the coefficients to create the filter. The Filter Design and Analysis System (FDAS) from Momentum Data Systems will even create assembly-language code to implement filter algorithms on a variety of DSP chips. Momentum president Jerry Purcell says the code-generation option for FDAS is very popular. "You can do a working program the minute you get the software," he says. "It gives you a big confidence boost and saves a week or two of trying to read through the manufacturer's manual to get a program running."

William Meshach, a consulting engineer for Loughborough Sound Images, recommends starting with C to learn DSP algorithms without the added confusion of real-time constraints. "Actually sit down with a setup where you can run something," he says. "Just write a C program that will record some stuff into memory, and then write a C program that scrambles it up somehow. Last of all, do simple math functions just to see how the waveforms are brought in, how they're stored, and how you manage the memory buffers."

Spectrum's Warren Cope believes you should familiarize yourself with DSP by taking three or four weeks to get a small practice design up and running, even if you have to borrow that time from your actual project. "I know that seems like a lot of time if you only have six months to do a design project," Cope says, but he stresses that you do eventually need to learn how to use a DSP chip and its tools, and you can get a jump on your project by first learning how to do all the parts of a design.


A filter-design software package can aid your introduction to DSP. The Filter Design and Analysis System (FDAS) from Momentum Data Systems helps you design and analyze filters and will even generate the program code needed to create the filter.


A DSP board can turn your PC into a prototype development system. This board from Spectrum Signal Processing contains Analog Devices' ADSP-2101 DSP chip, plus memory, analog I/O, and digital I/O.

By doing a small design first, you'll see where you need further help. You'll discover whether your assembler actually runs on your computer, whether your simulator runs in the available memory, and whether the simulator is fast enough to be useful. You can also find out if your compiler will compile and if your application library will link. You'll make sure everything works together and get some valuable experience on the way.

After writing a complete DSP program in C, you have to determine whether it's fast enough as is or if you need to rewrite the speed-demanding sections of code in assembly language. The simulators now available for DSP chips can help you determine which sections of code need additional work.

Before throwing any compiler-generated code out the window, though, make sure your expectations are realistic. Remember when you look at the code that a DSP chip is a RISC (reduced-instruction-set computer) processor. The code will look inefficient if you're used to looking at the assembly language for a CISC (complex-instruction-set computer) machine, but the microcode within a CISC machine hides many tasks that you have to explicitly tell a RISC machine to do.

You may find that C is helpful even if a state-of-the-art C compiler won't produce code that runs fast enough for the task you have at hand. Many companies use C as a development tool, even when they know their final product will contain only hand-generated assembly-language code; they use C to verify the functionality of their DSP algorithms, and then they turn to assembly language for speed.

Intelledex, a manufacturer of robots and vision systems in Corvallis, OR, is one company that takes this approach. The vision systems R&D group, headed by John McGarry, combines four AT&T DSP32C chips to make its algorithms run faster. McGarry explains that the group begins software development with a C compiler on a PC. In the next step, it uses AT&T's C cross-compiler to produce code for the DSP chips. The resulting program works, but is slow; a by-product, however, is an assembly-language listing that is helpful in understanding details of the program's operation. Next, the group writes a rough assembly-language version using the C code as a model. The final step is to optimize this program by taking advantage of the particular architectures of the DSP chips.

If you don't take advantage of your DSP chip's architecture, you won't get the speed the chip was designed for (see **box,** "Speed in the real world"). You have to understand the chip's parallel architecture to get the most out of your compiler. If you write code the way you always have, your compiler may not be able to optimize it. Every compiler has guidelines you should follow to get efficient code, so read the documentation carefully.

The way you use memory for data and program storage is one facet of DSP software design that can affect your program's execution speed. Many DSP chips have several separate memories that serve different purposes; if you use the memories improperly, your DSP chip won't work efficiently. For example, if you're multiplying two numbers together and you have to retrieve them both from the same memory, you'll need two sequential memory-read operations to fetch them. If, on the other hand, you have the numbers in separate memories, the processor can fetch them—as well as an instruction from a third memory—all at the same time. You must understand the memory-mapping features of the compiler and the architecture of the DSP chip to know where to place data for highest execution speed.

Another way to take full advantage of a DSP chip's speed is to use a DSP function library that's written in assembly language. Unlike C libraries, these libraries contain highly optimized assembly-language functions you call from C. A good one will give you the speed you need without making you write in assembly language.

Speed in the real world

A DSP program is a cross between a program for a supercomputer and a program for a real-time embedded system. The number-crunching algorithms that ran on supercomputers a few years ago now run on DSP chips and, in addition, must respond quickly to interrupts from real-world events. A DSP program is unacceptable if some part of it runs even a microsecond longer than the maximum allowable time.

Often you have to squeeze out every drop of performance to make your program fast enough. You can literally spend months shaving off instructions to increase speed. DSP chip architectures and instruction sets are designed to do some things—but not all things—very quickly. To get optimal performance, you must thoroughly understand your architecture and instruction set.

Speed is the only reason for using a DSP chip. Speed is also why programming a DSP chip is more difficult than programming a general-purpose μP. For a DSP chip to be fast, it has to perform many operations at the same time, so most DSP chips have a parallel architecture that makes their instruction sets very complex. A single instruction doesn't do just one thing; some operations of one instruction may even prepare for future operations. For example, while a DSP chip is multiplying two numbers from memory, the registers that point to the numbers can be incrementing to point to the next two numbers to be multiplied.

Keeping track of what a DSP chip is doing at any one time can be difficult. Because most of the chips are pipelined, a latency exists between an operation and the availability of the operation's results. If you're writing DSP programs in assembly language, you may have to wait a few cycles for results. Rather than wasting those cycles with no-ops, you can use instructions to accomplish something that may not even be related to the first instruction. The resulting code can get confusing. If you're writing in a high-level language, the assembly code produced by your compiler can be even more confusing.

Start your DSP experience by getting a small practice design up and running, even if you have to borrow the time from your actual project.

Analog Devices increases DSP speed by extending standard C. DSP/C is the company's implementation of an extended ANSI C standard being developed by the Numeric C Extensions Group, a working group of the ANSI X3J11 committee. In addition to increasing C's speed for DSP, DSP/C's nonproprietary extended features make DSP applications easier to write. "We believe the efficiency will be such that no one would ever be compelled to write an assembly-language subroutine or even in-line assembly code," says Kevin Leary, an Analog Devices staff engineer. Because DSP/C is a vector language instead of a scaler language, Leary continues, it can operate on vectors directly.

Debugging software for a DSP chip used to be a nightmare. There were no DSP debuggers or emulators on the market; you were on your own. The current availability of source-level debuggers for DSP chips is a major milestone. Without them, you would have to debug your program in assembly language and then try to figure out what the compiler did. This problem is even more serious with optimizing compilers, because the assembly code they produce looks so bizarre.

The ease of DSP programming depends on development tools and customer support. Texas Instruments currently leads the industry in the level of support and the number of tools it offers.

The first high-level-language source-level debugger, the TMS320C30 C Source Debugger from Texas Instruments, supports mixed C and assembly languages. You can look at the C code as it runs or at the assembly code as it runs. You can observe your application from a C perspective or an assembler perspective or even look at them both at the same time. If you have the assembly-language window open when you single step through the C code, the debugger highlights not only the line of C code, but also the assembly-language lines that implement the C line. You can display C variables by name, C structures, your link list (the list of all modules linked in your program), and other data. The debugger works on TI's XDS development system, EVM evaluation module, and C30 simulator.

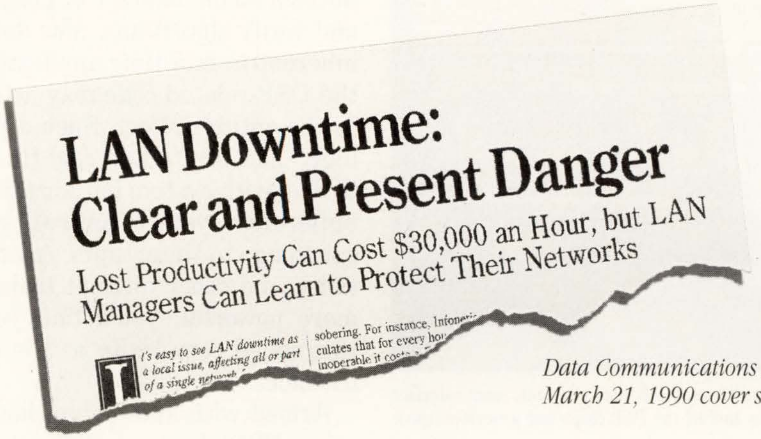
Another source-level debugger, the XDB from Intermetrics, will be available this fall, according to the company. The debugger and an optimizing C compiler will support the Motorola 96002. Otherwise similar to TI's debugger, the XDB doesn't use a mouse, and its windows aren't as convenient to use. It does have the same user interface as other debuggers from Intermetrics, so if you're using another μP as your system host, all your debugging tools can look the same. The Intermetrics tools run on a wide variety of computers.

When you develop and debug a program for a DSP chip, you'll run into the same types of problems you see in other real-time applications. Much of the advice you hear will also sound familiar. Analog Devices' Leary cautions against ignoring your runtime structure. He says his company's customers spend more time fixing runtime problems than anticipated.

You also need to understand the operation of on-chip peripherals. For example, many DSP chips have very capable, but complex, serial interfaces. Once you set up these interfaces, they're easy to use, but you have to learn how they operate in order to set them up.

Communications is another area that can cause trouble. Most of the time, a DSP chip does not work alone; complex systems have several DSP chips or at least a host controller. Designers often fail to examine bandwidth requirements for these systems, says Spectrum's Cope. "It can take a considerable amount of time for a DSP chip to send and receive data," he says. "A lot of people think they can just throw data to this other guy. They don't realize that [communication] kills a lot of your CPU cycles."

Your next embedded real-time system damn well better be bullet-proof.



LAN Downtime: Clear and Present Danger

Lost Productivity Can Cost $30,000 an Hour, but LAN Managers Can Learn to Protect Their Networks

It's easy to see LAN downtime as a local issue, affecting all or part of a single network

sobering. For instance, Infonetics calculates that for every hou inoperable it costs

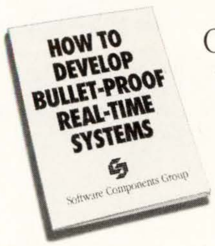
Data Communications Magazine
March 21, 1990 cover story

We're Software Components Group, and we deliver bullet-proof real-time technology to people who design embedded processor systems. One of our customers is SUN Microsystems, who uses our pSOS™ real-time operating system as the core of their new FDDI fibreoptic controller. Was bullet-proof important to their success?

"We had many engineers working to develop a super-reliable communications backbone that had to be faster than Ethernet and solid as a rock," said SUN project manager Bernie Mezrich. "What really sold us on pSOS was their PROBE™ debugger for the SUN environment. Without sophisticated debugging tools, we could have been killed by some very subtle problems."

From ATC to ATM to CAT to FDDI, our pSOS operating system has been proven in billions of dollars worth of mission-critical systems. No one else can deliver the technology you need to create a truly bullet-proof application.

Because no one else's kernel operating system, network handlers or file management components are as rock solid as pSOS. Because no one else offers you a fully integrated C or Ada environment, one that lets you develop and debug your multi-task, even multi-processor application at the source-code level. Over a network. On every major host—VAX, SUN, HP, PC. All of which is why you should take us up on our free offer.

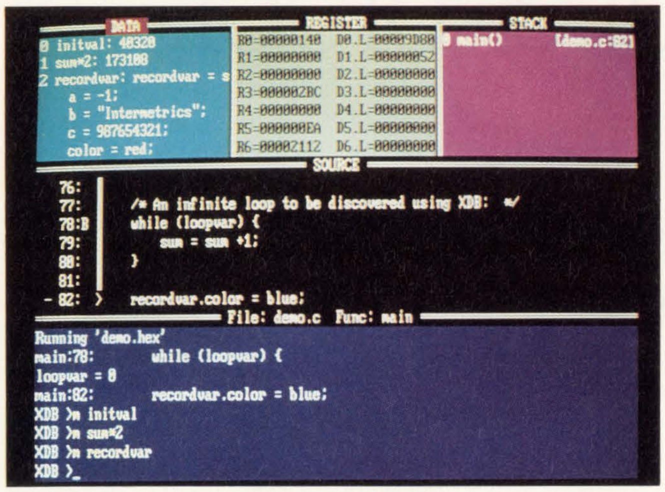

HOW TO DEVELOP BULLET-PROOF REAL-TIME SYSTEMS
Software Components Group

Call Software Components Group today for your copy of our white paper, *How to Develop Bullet-Proof Real-Time Systems.* Or risk costing your customers some serious money.

Telephone (800) 458-pSOS or FAX (408) 437-0711.

Software Components Group
BULLET-PROOF REAL-TIME TECHNOLOGY

Many companies use C to verify DSP algorithms' functionality and then turn to assembly language for speed.



```
DATA                              REGISTER                      STACK
0 inital: 48320               R0=00000140  D0.L=00009D80   0 main()      [demo.c:82]
1 sum*2: 173188               R1=00000000  D1.L=00000052
2 recordvar: recordvar = s    R2=00000000  D2.L=00000000
   a = -1;                    R3=000002BC  D3.L=00000000
   b = "Intermetrics";        R4=00000000  D4.L=00000000
   c = 987654321;             R5=000000EA  D5.L=00000000
   color = red;               R6=00002112  D6.L=00000000
                              SOURCE
   76:
   77:      /* An infinite loop to be discovered using XDB:  */
   78:B      while (loopvar) {
   79:         sum = sum +1;
   80:      }
   81:
 - 82: >    recordvar.color = blue;
                      File: demo.c  Func: main
Running 'demo.hex'
main:78:        while (loopvar) {
loopvar = 0
main:82:        recordvar.color = blue;
XDB >m inital
XDB >m sum*2
XDB >m recordvar
XDB >_
```

A source-level debugger from Intermetrics, XDB uses the same user interface for all the host computers it runs on and all the DSP chips and general-purpose µPs it supports.

Your system design affects the difficulty of implementing the overall program. If your DSP chip is a coprocessor to another µP, you might be able to avoid real-time interrupt handling, sampling data at exact intervals, or interlacing your algorithms with real-time control of your system. A coprocessor usually accepts data from the host µP, then processes it and flags the host when it's done; there are few surprises. You can debug your program simply by feeding it known data and verifying the results.

A stand-alone application, on the other hand, produces a whole new set of problems. You have to create and verify algorithms, plus deal with all the difficulties inherent to real-time applications. The effort to create the DSP-related code may actually be minor compared to the entire project. Such a project may look surprisingly like any other real-time application. You have to deal with system initialization, data acquisition, data buffering, system control, running algorithms, responding to interrupts, memory-management functions, and other difficult tasks. As DSP chips become more powerful, you'll find your programs taking on this added complexity as the need for a separate host µP fades.

Armed with knowledge, however, you shouldn't find using DSP chips too difficult. The increasing number of development tools helps, especially if you're already familiar with similar tools for general-purpose µPs. The continued evolution of DSP chips makes the job easier. And the ever increasing number of function libraries keeps you from reinventing the wheel. EDN

Article Interest Quotient (Circle One)
High 476 Medium 477 Low 478

For more information . . .

For more information on DSP chips and development tools discussed in this article, circle the appropriate numbers on the Information Retrieval Service card, or use EDN's Express Request service. When you contact the manufacturers directly, please let them know you saw their products in EDN.

Analog Devices Inc
1 Technology Way
Norwood, MA 02062
(617) 461-3074
Circle No. 650

AT&T Microelectronics
Dept 52AL300240
555 Union Blvd
Allentown, PA 18103
(800) 372-2447;
in Canada, (800) 553-2448
Circle No. 651

Intermetrics Inc
733 Concord Ave
Cambridge, MA 02138
(800) 356-3594;
in MA, (617) 661-0072
Circle No. 652

Momentum Data Systems
1520 Nutmeg Pl, Suite 108
Costa Mesa, CA 92626
(714) 557-6884
Circle No. 653

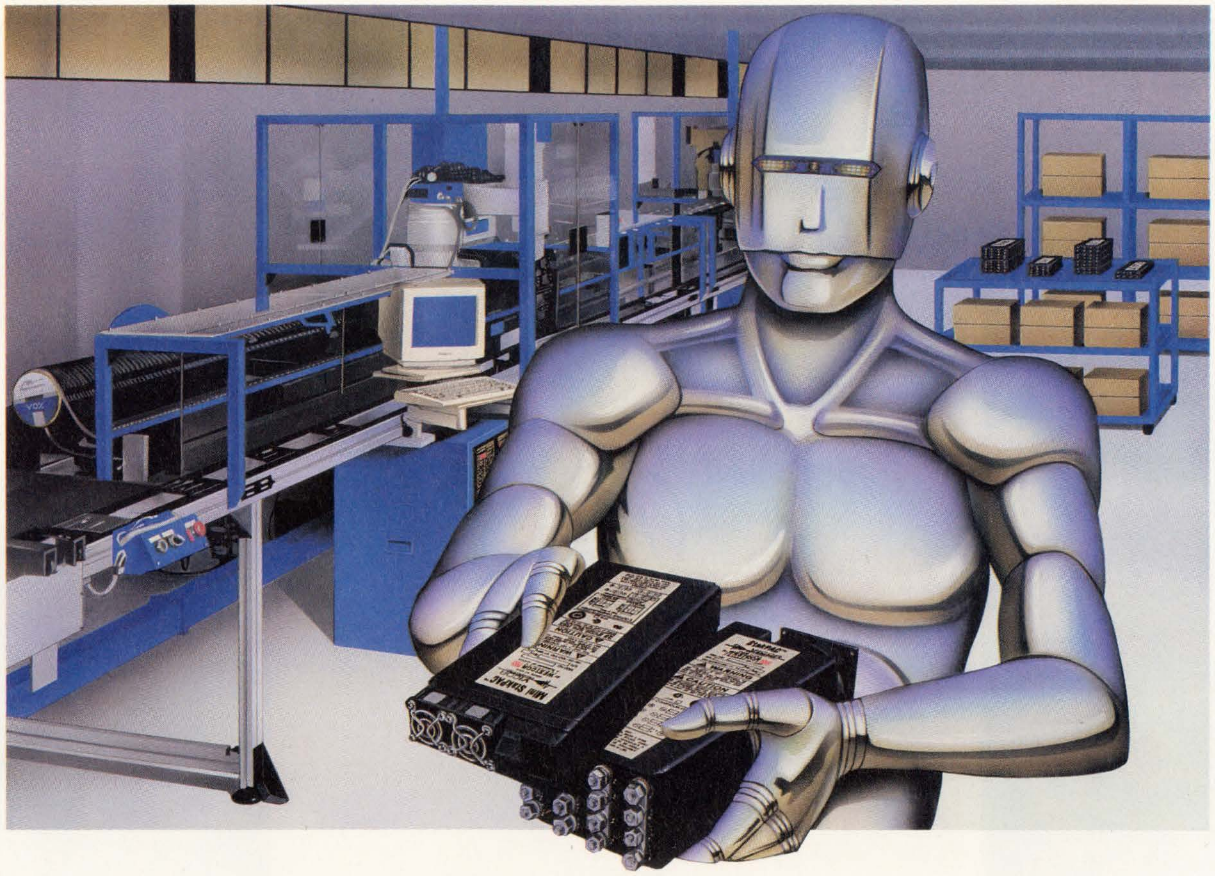
Motorola Inc
Microprocessor Products Group
6501 William Cannon Dr W
Austin, TX 78735
(512) 891-2030
Circle No. 654

Sonitech International Inc
14 Mica Lane, Suite 208
Wellesley, MA 02181
(617) 235-6824
Circle No. 655

Spectron Microsystems Inc
600 Ward Dr
Santa Barbara, CA 93111
(805) 967-0503
Circle No. 656

Spectrum Signal Processing Inc
Box 8110-25
Blaine, WA 98230
(800) 663-8986;
in WA, (604) 438-7266
Circle No. 657

Texas Instruments Inc
Semiconductor Group, SC-9026
Box 809066
Dallas, TX 75380
(800) 232-3200, ext 700
Circle No. 658



RoboPower

STAKPAC™		MINI STAKPAC™
1200 Watts	**Power**	600 Watts
110/220 VAC	**Input**	110/220 VAC
Up to 8	**Outputs**	Up to 5
3.2"x5.5"x11.5	**Dimensions**	1.9"x5.5"x12"
Fan-Cooled	**Cooling**	Twin Fans

Each StakPAC output is factory configured utilizing Vicor's robotically manufactured power converters...VI-200 series modules. Consider the advantages of a StakPAC customized for your system needs with automized power modules:

USER DEFINABLE OUTPUTS— The use of proven standard catalog modules offers the features of a custom without the associated risk or investment.

STANDARD MODELS—Many preconfigured standards available.

QUICK DELIVERY—Typical delivery 1 week or less for custom or standard evaluation units.

COMPACTNESS—Low profile packages provide up to 6 watts/cubic inch, twice the industry norm.

UL, CSA, TUV SAFETY AGENCY APPROVAL—All StakPAC configurations are approved, standard or custom.

EMI—FCC/VDE Level A, conducted.

StakPACs are designed and built by Westcor Corporation, Los Gatos, CA, a Vicor subsidiary. StakPACs are sold world-wide through Vicor Corporation, Andover, MA.

STAKPAC STANDARDS 1200 WATT MODELS

Model	Output Voltage (VDC) and Maximum Current (amperes) per Channel				
	#1	#2	#3	#4	#5
Single Output					
SP1-1801	2 @ 240				
SP1-1802	5 @ 240				
SP1-1603	12 @ 100				
SP1-1604	15 @ 80				
SP1-1605	24 @ 50				
SP1-1606	28 @ 42				
SP1-1607	48 @ 25				
Dual Output					
SP2-1801	2 @ 120	5 @ 120			
SP2-1802	5 @ 120	5 @ 120			
SP2-1803	5 @ 120	12 @ 66			
SP2-1804	12 @ 66	12 @ 66			
SP2-1805	15 @ 53	15 @ 53			
Triple Output					
SP3-1801	5 @ 180	12 @ 16	12 @ 16		
SP3-1802	5 @ 150	12 @ 33	12 @ 16		
SP3-1803	5 @ 180	15 @ 13	15 @ 13		
SP3-1804	5 @ 150	15 @ 26	15 @ 13		
Quad Output					
SP4-1801	5 @ 150	12 @ 16	12 @ 16	5 @ 30	
SP4-1802	5 @ 150	15 @ 13	15 @ 13	5 @ 30	
SP4-1803	5 @ 150	12 @ 16	12 @ 16	24 @ 8	
SP4-1804	5 @ 150	15 @ 13	15 @ 13	24 @ 8	
Five Output					
SP5-1801	5 @ 120	12 @ 16	12 @ 16	5 @ 30	24 @ 8
SP5-1802	5 @ 120	15 @ 13	15 @ 13	5 @ 30	24 @ 8
Seven Output					
SP7-1801	5 @ 60	12 @ 16	12 @ 16	24 @ 8	24 @ 8
	#6	#7			
	5.2 @ 28	2 @ 30			

Total output power may not exceed 1200* watts for any model, single or multiple output. Lower power StakPAC models and many other configurations are available.
*Standard models supply 1100 watts; high-powered version 1200 watts. Please contact the factory.

MINI STAKPAC STANDARDS 600 WATT MODELS

Model	Output Voltage (VDC) and Maximum Current (amperes) per Channel				
	#1	#2	#3	#4	#5
Single Output					
ST1-1401	2 @ 120				
ST1-1402	5 @ 120				
ST1-1301	12 @ 50				
ST1-1302	15 @ 40				
ST1-1303	24 @ 25				
ST1-1304	28 @ 21				
ST1-1305	48 @ 13				
Dual Output					
ST2-1401	2 @ 60	5 @ 60			
ST2-1402	5 @ 60	5 @ 60			
ST2-1403	5 @ 60	12 @ 33			
ST2-1404	12 @ 33	12 @ 33			
ST2-1405	15 @ 26	15 @ 26			
Triple Output					
ST3-1401	5 @ 60	12 @ 16	12 @ 16		
ST3-1402	5 @ 60	15 @ 13	15 @ 13		
ST3-1501	5 @ 90	12 @ 8	12 @ 8		
Quad Output					
ST4-1401	5 @ 30	12 @ 16	12 @ 16	5 @ 30	
ST4-1402	5 @ 30	15 @ 13	15 @ 13	5 @ 30	
ST4-1403	5 @ 30	12 @ 16	12 @ 16	24 @ 8	
ST4-1501	5 @ 30	15 @ 13	15 @ 13	24 @ 8	
ST4-1502	5 @ 60	12 @ 16	12 @ 8	5 @ 15	
ST4-1503	5 @ 60	15 @ 13	15 @ 7	5 @ 15	
ST4-1504	5 @ 60	12 @ 16	12 @ 8	24 @ 4	
ST4-1505	5 @ 60	15 @ 13	15 @ 7	24 @ 4	
Five Output					
ST5-1501	5 @ 30	12 @ 16	12 @ 16	5 @ 15	24 @ 4
ST5-1502	5 @ 30	15 @ 13	15 @ 13	5 @ 15	24 @ 4

Total output power may not exceed 600 watts for any model, single or multiple output. Lower power Mini StakPAC models and many other configurations are available. Please contact the factory.

For ordering information call Vicor Express at 1-800-735-6200 or (508) 470-2900 at ext. 265.

For technical information contact Westcor at (408) 395-7050 or FAX (408) 395-1518 or call Vicor.

TM

VICOR CORPORATION

Common Stock Traded on NASDAQ under "VICR"

WESTCOR CORPORATION
485-100 Alberto Way
Los Gatos, CA 95032

VICOR CORPORATION
23 Frontage Road
Andover, MA 01810

Motorola made us do it.

They made us an offer we couldn't refuse. They wanted our ES 1800 development system to support their new 68302 Integrated Multiprotocol Processor. Naturally, we jumped at the offer.

Which gives you a big jump on your competition. Because now you can put the ES 1800 to work debugging your design today, instead of waiting months for other development tools to roll out.

And the ES 1800 system flies at maximum-rated clock speeds. With a full set of tools to support the 68302's special features, including three serial controllers, seven DMA channels, and chip selects. Plus complete software development tools and host support.

What's more, we offer training, installation, and application assistance. All to save you time and money. All to bring your design to market faster. Just as Applied Microsystems has done with more than 12,000 systems worldwide.

For a free demonstration, call **1-800-343-3659** (In WA, (206) 882-2000). Ask for Telemarketing.

Oh, and don't forget to thank Motorola.

Applied Microsystems Corporation

For the name of your nearest distributor in Europe, call 44-(0)-296-625462. Europe Fax 44-296-623460. Or contact Applied Microsystems Corporation, Ltd., Chiltern Court, High Street, Wendover, Aylesbury, Bucks, HP22 6EP, United Kingdom. **In Japan, call 03-493-0770.** Japan Fax 03-493-7270. Or contact Applied Microsystems Japan, Ltd., Nihon Seimei, Nishi-Gotanda Building, 7-24-5 Nishi-Gotanda, Shinagawa-KU, Tokyo T141, Japan.

Debugging guidelines facilitate software development

Although software debugging can be something of a black art, some general rules do apply. Consider the various debugging options that are available at different stages of software design and integration.

Andy Lantz, *Intermetrics Inc*

Programmers, unfortunately, are not free from the human trait of fallibility. During every software project, engineers stare at listings and screens and wonder why their programs don't work. Although bugs, like human fallibility, are here to stay, finding them is easier than it used to be. Improved debugging tools help, as does experience. Collective experience yields some general guidelines that can help make debugging less of an art and more of a science. These guidelines include holding design meetings and code walkthroughs; using modular design; and taking full advantage of simulators, emulators, and ROM monitors.

One of the best ways to simplify software debugging is to use structured design methods. The design and coding phases of a project are not often thought of as steps in the debugging process, but the work done during these phases can have the most significant impact on the amount of time spent later in debugging. Top-down design, a rigorous review process, and adherence to strict coding standards and other principles of structured software development can make life much easier during a project's debugging phase.

Design reviews and a code walkthrough are crucial to the schedule of any significant software project. At the very least, your project should have a top-level design review, a detailed design review, and a code walkthrough. All can be instrumental in uncovering problems.

The top-level design review is an opportunity to find bugs at the conceptual level of a project's design. A program's designers should produce a top-level-design document detailing each of the program's functional units and the relationships and data flow between those units. This document should be distributed to a group of reviewers—experienced software developers not directly involved with the application's development—before the design review meeting. At the meeting, the reviewers should play devil's advocate and try to poke holes in the design.

The detailed design review and the code walkthrough should use the same forum as the top-level review. At the detailed review, reviewers examine the detailed design document, which contains pseudocode for every function that will be in the program; at the code walkthrough, reviewers examine the final code implementation.

A top-level design specifies ideas, functions, and concepts. At some point, you define what modules you will create to implement these concepts, then you design the modules. Modular design is not just good software engineering practice—it can also prevent some kinds of bugs from occurring in your code. Suppose you see the same sequence of statements appearing in multiple places in your program. If you incorporate that block of code into its own function or macro, you accomplish two objectives. First, you eliminate the possibility that the code is correct in some places and not in others. Second, if you need to change this common code, you have to change only one instance of it, thereby reducing the chance of clerical error.

C is efficient for building pointers and generating pointer bugs of the worst magnitude.

The way you use C—the high-level language of choice for coding embedded-systems applications—also greatly affects later debugging. C offers programmers many ways to get around cumbersome type restrictions found in stricter languages such as Pascal and Ada, but this flexibility can be a double-edged sword. Just as you can write C code that rivals assembly language's efficiency in handling pointers, you can also quickly and efficiently create insidious stray-pointer bugs of the worst magnitude. Fortunately, you can use several techniques to stringently check C code at compile time.

The *lint* program is probably the single most useful compile-time aid for preventing C bugs. Originally supplied as a utility program with Unix but now available elsewhere, *lint* accepts a C program as input and warns of unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose values are constant. The program also finds functions that return values in some places and not in others, functions called with varying numbers of arguments, and functions whose values are not used.

Lint sounds great, yet many people categorically refuse to use it, complaining that it forces them to wade through countless pages of warnings that aren't helpful or appropriate. These people may not have experimented with the various switches lint offers to disable checking that is irrelevant or redundant.

Even if *lint* isn't available on your system, there are other ways to get compile-time warnings of potential bugs. One way is to use a compiler that warns of situations such as possible infinite loops or local variables that your code never references. Another way to get compile-time warnings is to use a compiler that supports ANSI C's function prototype feature **(Fig 1)**.

Function prototypes were added to the ANSI standard for C so that programmers could make compilers perform stronger type checking on statements that declare, define, or call functions. A function prototype is a variant form of a function declaration; the difference is that a function declaration only names the function and its return type, but a function prototype includes information on the number and types of parameters the function requires. If a programmer uses a function prototype instead of a function declaration, the compiler will check to see if the corresponding function definition and any calls to the function have the proper number and types of parameters.

Other good coding habits can also help you avoid common C pitfalls. If you're comparing strings, for example, it's much safer to define a preprocessor macro called STRING_EQ than to risk forgetting that the *!* in *!(strcmp(x,y))* is required to test for equality between string x and string y.

In general, using preprocessor macros makes for good design practice. Don't use numeric literals throughout your program; there are too many opportunities for typos. Instead, define preprocessor names for them (for example, "#define MAXLEN 256"). If you type a numeric literal incorrectly, it may cause problems that you won't detect until the program mysteriously fails at runtime. If you type a macro name incorrectly, however, the compiler will tell you right away that you're referencing an unknown name.

Whenever you have a test for equality between a constant and an expression, get into the habit of putting the constant first: *if (7 == x)*. This practice makes it impossible for you to commit one of the most common and subtle C programming errors: using = when you mean to use == for a comparison. The compiler will not accept *if (7 = x)*, but it will accept *if (x = 7)* with no complaints. If you're trying to compare x with 7, the latter expression will not evaluate the way you want it to.

Print statements provide flexibility

Perhaps the most common debugging technique used at the coding stage of native applications (applications in which the code compiles and runs on the same system) is inserting *printf()* calls into a program. At runtime, these statements cause the program to display values of important variables and structures. Using this technique for embedded systems is complicated because the system might not have an obvious destination for standard output (*stdout*). Even if your system doesn't have standard I/O, you have other options, such as complex breakpoints and simulated I/O, for displaying diagnostic messages.

If you can use *printf()* calls for diagnostics, you'll benefit from enclosing the debugging code in conditional compilation directives. For example, define a preprocessor variable called DEBUG, which you can turn on or off at compile time. Then, you can isolate your debugging code as in the following:

```
#if DEBUG
printf("value of index is %d\n", index);
.
.
#endif
```

```
/* non-prototype function declaration for f() */
double f();

/* non-prototype function definition for f() */
        double
f(a,b,c)
        char a;
        int b;
        double c;
{
        return (a + b + c);
}
```
(a)

```
/* function prototype declaration for f() */
double f(char a, int b, double c);

/* function prototype definition for f() */
        double
f(char a, int b, double c)
{
        return (a + b + c);
}
```
(b)

```
Compiling the following  code...

/* function prototype declaration for f() */
double f(char a, int b, double c);

/* incorrect function definition for f() */
        double
f(char a, double b, int c)
{
        return (a + b + c);
}

...results in the following error messages

:FE:badproto.c:6:       f: Symbol redefined
:FE:badproto.c:6:       First declared type assumed for function
```
(c)

```
Compiling the following  code...

/* function prototype declaration for f() */
double f(char a, int b, double c);

/* function definition for f() */
        double
f(char a, int b, double c)
{
        return (a + b + c);
}

main()
{
        double  g;

        g = f('a', 3.14159);
}


...results in the following error message:


:FE:badproto2.c:15:    Mismatch between # of args at call and function
prototype
```
(d)

Fig 1—Function prototypes help avoid some C bugs by causing stronger type checking at compile time. The code shown in (a) contains a function declaration and a corresponding function definition; the code in (b) is the same function, using function prototypes for the declaration and the definition. If you use function prototypes instead of ordinary function declarations, your compiler will check for the correct number and types of function arguments. For example, (c) shows a compiler error message resulting from a type mismatch between a function prototype declaration and the corresponding function declaration. The code in (d) shows the compiler error message resulting from an incorrect call to a prototype function.

Develop programs top-down; debug them bottom-up.

Obvious locations for these kinds of messages are at the beginnings and ends of functions and inside loops.

The listings your compiler and linker produce can be a big help both before and after you start debugging. Many compilers can produce a listing that shows the user's original C code interleaved with the assembly language that corresponds to the generated machine instructions (**Fig 2**). These listings can alert you to bugs in your program.

Suppose, for example, that you've written a block of code that an optimizing compiler determines to be unreachable. The conspicuous lack of generated instructions for that source code is readily apparent during a casual glance through an interleaved listing. You can use a global symbol map to find similar anomalies in your program. A segment showing a length of zero can be an indication that the program has been written or linked incorrectly.

Decide where to run your code

After you've written most or all of your software, you still have to decide where it will run when you debug it. Your choice will depend on the readiness of your hardware and on your budget. Simulators let you debug software before the hardware is ready; emulators and ROM monitors let you integrate and test your hardware in conjunction with the software.

If the software engineers are ahead of the hardware engineers and your target board isn't yet ready for testing, you'll probably want to begin debugging your software by determining whether it is algorithmically correct. The best way to make this determination is with simulation. The principle behind simulation is that in the preliminary phases of debugging an embedded system, some aspects of the software can be tested independently of the target hardware.

A choice of simulators

You can choose either a software instruction-set simulator or a hardware simulation board. An instruction-set simulator is software that runs on the development host and mimics the target microprocessor. A hardware simulation board is a single-board computer that has a microprocessor, ROM containing a simple control program, and some RAM into which to download the application program. The simulation board usually connects to the host via a serial connection or, if the host is an IBM PC or compatible computer, via the AT bus.

Many simulators can be controlled by a source-level debugger, which lets you debug your code at the source-code level rather than at the assembly-language level. For the C programmer, source-level debugging is a requirement for maximum productivity.

Once you have an execution environment for your program, the next step is to organize your debugging strategy. Your first tendency might be to try running the program in its entirety. Very likely, you will encounter problems. At the discovery of the first bug, programmers often employ some kind of binary search, using breakpoints to iteratively narrow down the bug's location. Although this approach is popular, by itself it is not always as efficient as the more structured techniques of bottom-up debugging.

The best way to debug software is the opposite of the way you developed it. Develop programs starting at the highest level of organization and add detail during subsequent stages of development. Use bottom-up debugging to ensure that low-level functions are working correctly before you examine the behavior of your program as a whole. Observing the behavior of an individual function in the context of bottom-up debugging is known as unit testing.

```
A -->   * Variable i is in the D5 Register
        * Variable j is in the D4 Register
        * Variable k is in the D3 Register
                  .
                  .
                  .
        *5           for (i = 1; i < 10; ++i) {

                  MOVEQ.L #1,D5
B -->   *(code hoisted from following statement)
                  MOVE    D3,D2
                  MULS    D4,D2

        L20001
        *6                f(i + j * k);
                  MOVE    D5,D1
                  ADD     D2,D1
                  MOVE    D1,-(A7)
                  JSR     _f

C -->   *(see line 5)
                  ADDQ.L  #2,A7
                  ADDQ    #1,D5
                  CMPI    #10,D5
                  BLT.S   L20001
                  .
                  .
                  .
```

Fig 2—An interleaved listing can be helpful in debugging, showing how a compiler generates assembly-language code from lines of C code. The listing here shows how the compiler allocates local variable in registers (a), how the compiler generates code for a loop-invariant expression before the code for the body of the loop (b), and how the code for the increment-test-and-branch part of the loop is separated from the beginning of the loop (c).

There are a couple of ways to perform unit testing. The oldest method is to write a driver program for a routine. A driver is a routine that does little else than call a single function. Debugging a program consisting solely of a single function and its driver would be ideal, but isolating a single function in this way is often difficult: There are just too many interdependencies to avoid linking in a number of other functions. A much easier way of doing unit testing is to have a source-level debugger do the work for you. If your debugger can evaluate C expressions containing function calls, you can test a single function by passing all kinds of hypothetical parameters to it.

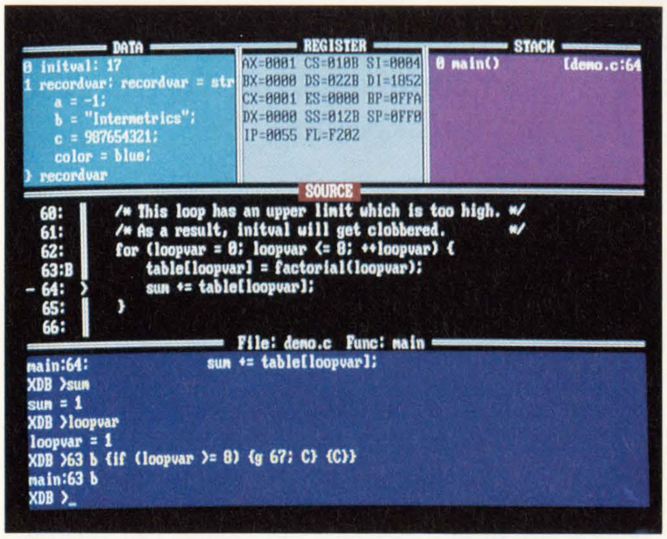
Emulator or ROM monitor?

Once your target hardware is available, you'll have more choices for where your code runs during debugging. Both in-circuit emulators and ROM-based monitors let you test your software along with the hardware. No matter what combination of hardware you choose to assist your debugging, you should try to exploit your investment as fully as possible. Learn as much as you can about the advanced features of your debugger's command language.

One of the most useful features of an in-circuit emulator is, of course, its ability to set breakpoints that impose no penalty in real-time execution speed. A code breakpoint instructs the emulator to halt execution when an instruction at a particular address executes. Code breakpoints are one of the mainstays of day-to-day debugging practice.

But programmers often overlook data breakpoints, which cause a program to halt whenever it reads from or writes to a specified memory address. Suppose you have a variable that is getting clobbered at some unknown line in your program. Rather than trying to discover the offending line by setting code breakpoints and inspecting the variable's value at various statements, use a data breakpoint to check for write operations to the variable's address.

Sometimes your requirements are more specific: You want your program to halt only if one particular value is being written to a certain memory location. Some emulators answer this need by offering data breakpoints with an additional capability: You can specify not only the breakpoint address, but also the one write-data value that you want to result in an actual stopping of the program. This capability can help you track down errors that occur at boundary conditions for input-data values. Data breakpoints can even operate over ranges of addresses. When applied over a range, they can help determine where a program attempts to write to ROM.



```
DATA
0 initval: 17
1 recordvar: recordvar = str
    a = -1;
    b = "Intermetrics";
    c = 987654321;
    color = blue;
} recordvar

REGISTER
AX=0001 CS=010B SI=0004
BX=0000 DS=022B DI=1852
CX=0001 ES=0000 BP=0FFA
DX=0000 SS=012B SP=0FF0
IP=0055 FL=F202

STACK
0 main()              [demo.c:64

SOURCE
  60:  |    /* This loop has an upper limit which is too high. */
  61:  |    /* As a result, initval will get clobbered.         */
  62:  |    for (loopvar = 0; loopvar <= 8; ++loopvar) {
  63:B |        table[loopvar] = factorial(loopvar);
- 64:  >        sum += table[loopvar];
  65:  |    }
  66:  |

File: demo.c  Func: main
main:64:                sum += table[loopvar];
XDB >sum
sum = 1
XDB >loopvar
loopvar = 1
XDB >63 b {if (loopvar >= 8) {g 67; C} {C}}
main:63 b
XDB >_
```

Fig 3—Complex breakpoints, which are supported by some debuggers, let you test fixes to your program before you change the source code. This example on the XDB debugger from Intermetrics (Cambridge, MA) shows a simulated patch. The user has set a breakpoint at line 63, which causes the program to exit the *for* loop (using the commands "g 67" to go to line 67 and "C" to continue execution) when the value of the variable ***loopvar*** becomes greater than or equal to 8. This patch temporarily fixes a bug caused by an incorrect upper limit in the comparison part of the loop.

Source-level debuggers offer one very obvious advantage over other debuggers. You can set a breakpoint by referring to a source-level statement's line number or by referring to a variable name; you don't have to refer to hexadecimal memory addresses. But few people make the best use of their source-level debugger's other features. They rely on three or four basic commands, forgetting that they can attack some of their problems more effectively with others.

For example, some source-level debuggers allow conditional breakpoints. These debuggers will stop your program at such a breakpoint and then check a simple condition to determine if the program will continue or remain at the breakpoint.

Other source-level debuggers provide complex breakpoints—breakpoints with attached command lists (**Fig 3**). The commands, which can be debugger instructions or C expressions, are performed when the breakpoint is hit. You can exploit this feature in a variety of ways. In the simplest case, you can use commands to display the value of a variable or an expression every time your program reaches a particular line of code.

Use complex breakpoints for patches

One powerful use of complex breakpoints is for simulating code patches. In the simplest case, suppose you need to add a missing statement to your code. You can set a complex breakpoint on the statement after the missing statement, append the missing statement to the breakpoint, and follow it with a command to continue execution.

Complex breakpoints let you attach lists of debugger commands or C instructions to ordinary breakpoints.

Deleting a line of code using complex breakpoints is a little more risky. You set a breakpoint on the statement to be deleted and then append a list of commands to the breakpoint. One of the commands changes the value of the program counter register to the address of the statement after the one to be deleted; another is a command to continue.

There are several reasons you should use this approach with caution. First, you can inadvertently cause condition codes to have the wrong values because a deleted line of code no longer executes and therefore doesn't set the codes. If program control later passes to a part of the program that evaluates such condition codes, then the result will be unpredictable. Also, deleting instructions might disrupt the stack pointer. Nevertheless, this kind of patching is valuable because it lets you test the logic of potential fixes without recompiling your code.

The debugging technique of placing *printf()* statements throughout your code is a long-used method of getting diagnostic information about the values of variables at runtime. But with embedded systems, as opposed to native application programs, the output—if the system has any output at all—does not necessarily go to a screen. Sometimes all that happens is that values in certain memory addresses change. One solution to this problem involves using a debugger's simulated I/O capability. With simulated I/O, you don't have to use peripheral hardware for input and output; you can use the resources of the host computer, such as disk files, the keyboard, or the screen. Many debugger packages offering simulated I/O include alternate versions of the compiler's standard I/O library routines.

In the course of tracking down an especially elusive bug, it is not uncommon for several engineers to work in shifts. By using the debugger to record into a disk file all the debugging commands and results during a particular shift, you can ensure that the next shift sees what has transpired. The next user simply uses the debugger to read in the file from the previous debugging session.

Many source-level debuggers also offer on-screen windows for monitoring variables and data expressions, automatically displaying the values whenever execution stops. Remembering to use this feature can save you from typing a command to display these values at each breakpoint. If you find yourself repeatedly asking the debugger to display the value of a particular expression after single steps or breakpoints, you should monitor the expression's value in a window.

Even with a source-level debugger's powerful arsenal of commands for high-level debugging, at times you'll need access to the low-level features of your emulator's own command set (**Fig 4**). These low-level features let you observe bus activity, establish trigger conditions for tracing, and set any type of emulator-specific breakpoints that is not available in the debugger itself. Debugging at the integration and testing phases of a project should include a balance of high-and low-level debugging. Most source-level debuggers let you "drop down" to the emulator's interface during a debugging session and then return to the high level.

Sometimes, however, an in-circuit emulator isn't the best choice for debugging, or an emulator might not be available for a chip. One debugging approach circumvents this problem and is rapidly growing in popularity. In the technique sometimes called remote debugging, a source-level debugger communicates directly with a monitor program burned into PROMs on the user's actual target board.

Like emulators, some ROM monitors can accumulate a trace buffer of a program's execution history in RAM. Because one CPU runs both the ROM monitor and the user's application code, ROM monitors can't save trace information without stopping execution before each machine instruction. ROM monitors make up for this sacrifice of real-time tracing by providing extended information in their trace buffers. Suppliers of ROM monitors reason that if a monitor has to run a program in step mode to get trace information, it might as well record more information than just the addresses of executed instructions.

With some ROM monitors, you have the option of saving only the program counter values in the trace buffer or saving both those values and the values of each instruction's operands before and after instruction execution. This trace with full data movement can pinpoint the precise instruction that causes a data value—even in a register—to go bad. Because this kind of trace also shows the effective addresses computed for instructions, you can use it to detect errors in assembly-language code.

Another difference between emulators and ROM monitors involves the ability to set breakpoints on data accesses. Because emulators watch external buses to detect accesses to particular memory locations, they can detect these kinds of breakpoints in real time. A ROM monitor can't detect an access to a memory location; it can, however, check the contents of a memory address before each instruction to see if the value has changed. A ROM monitor may let you set up a breakpoint that compares the contents of a register with some mask or value before each instruction and halts when a particular condition is met. Because an emulator cannot interrogate the values of internal registers while the processor is running, it cannot use its hardware to set breakpoints on register accesses or changes. As with an execution trace, a ROM monitor—resigned to running in trace mode—tries to compensate for not being able to set breakpoints by providing additional information not available to an emulator.

ROM monitors can also be useful for troubleshooting finished, but malfunctioning, products. If you may have occasion to debug your product after it has been sold, leaving a ROM monitor in the final version of your board gives you the opportunity to debug in the field. Keeping the monitor on the board adds to the cost and ROM requirements of your product, but the benefits may far outweigh the costs.

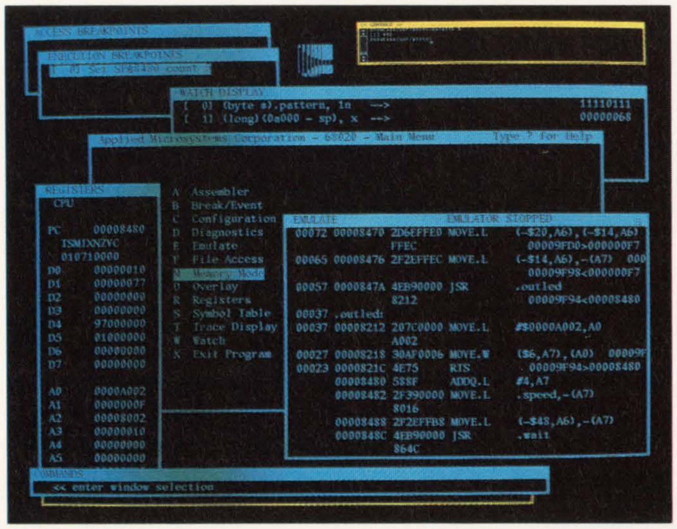
Embedded systems, extra problems

Native applications, of course, tend to be easier to debug than embedded systems. Developers of native applications programs don't have to know exactly where their code will reside in memory, for example. They also tend to have few worries about interrupts and external I/O devices. Embedded-systems programmers are not so fortunate.

Unlike native-system programs, which the host's operating system loads into available system memory, an embedded-system program must go into explicitly specified target-hardware addresses. One of the most troublesome problems that can result from the incorrect location of your program is the allocation of too small a region in target memory for the runtime stack. To further complicate matters, the symptoms of this problem are subtle. The program may mysteriously crash only after several iterations of a recursive routine or in a deep nesting of other function calls. Checking the value of your program's stack pointer at strategic places in your code and comparing its value with addresses in your global symbol map is a good practice.

Linking your program with a runtime library that is in some way inappropriate for your application may also result in bugs that are extremely hard to track down. Take great care when you build your program to link only with runtime libraries that have been proven correct and that match your compiler's calling conventions.

Debugging interrupt-driven code is one of the chal-



```
ACCESS BREAKPOINTS

EXECUTION BREAKPOINTS
[  0] Set SP@8480 count

WATCH DISPLAY
[  0] (byte s).pattern, in    -->                              11110111
[  1] (long)(0a000 - sp), x   -->                              00000068

Applied Microsystems Corporation - 68020 - Main Menu     Type ? For Help

REGISTERS
 CPU

PC       00008480
  ISMIXNZVC
  010710000
D0       00000010
D1       00000077
D2       00000000
D3       00000000
D4       97000000
D5       01000000
D6       00000080
D7       00000000

A0       0000A002
A1       0000000F
A2       00008002
A3       00000010
A4       00000000
A5       00000000

A  Assembler
B  Break/Event
C  Configuration
D  Diagnostics
E  Emulate
F  File Access
M  Memory Mode
O  Overlay
R  Registers
S  Symbol Table
T  Trace Display
W  Watch
X  Exit Program

EMULATE                          EMULATOR STOPPED
00072 00008470 2D6EFFE0 MOVE.L   (-$20,A6),(-$14,A6)
               FFEC                00009FD0>000000F7
00065 00008476 2F2EFFEC MOVE.L   (-$14,A6),-(A7)  000
                                   00009F98<000000F7
00057 0000847A 4EB90000 JSR      .outled
               8212                00009F94<00008480
00037 .outled:
00037 00008212 207C0000 MOVE.L   #$0000A002,A0
               A002
00027 00008218 30AF0006 MOVE.W   ($6,A7),(A0)  00009F
00023 0000821C 4E75     RTS        00009F94>00008480
      00008480 588F     ADDQ.L   #4,A7
      00008482 2F390000 MOVE.L   .speed,-(A7)
               8016
      00008488 2F2EFFB8 MOVE.L   (-$48,A6),-(A7)
      0000848C 4EB90000 JSR      .wait
               864C

COMMANDS
<< enter window selection
```

Fig 4—An emulator's user interface allows in-depth low-level debugging. This display from the interface to Applied Microsystems' (Redmond, WA) EL 3200 emulator shows windows for registers, breakpoints, the instruction trace, and monitored data.

lenges that embedded-systems applications present. A trace buffer for an interrupt-driven program frequently contains interrupt-handler code interspersed with code from the nonhandler routines. Looking at the complete asynchronous trace may be useful in some instances, but a more limited, qualified trace can also be helpful. You should make the best use of any trace-qualifying mechanisms you can.

Many emulators let you specify trigger conditions that determine when executed instructions will be recorded in the trace buffer. Some ROM monitors let you use conditional breakpoints to enable and disable tracing. If you're trying to debug an individual interrupt handler, you may want to give the appropriate debugger commands so that tracing is initiated at the beginning of the handler and disabled upon exit. Conversely, once a particular handler is known to work properly, you can arrange for the trace to be disabled on entry to the handler and enabled on exit. EDN

Author's biography

Andy Lantz has been debugging software for a variety of applications for more than eight years. He is a senior applications engineer at Intermetrics Inc (Cambridge, MA) where he provides technical support and training to software engineers who design embedded systems. Andy holds a bachelor's degree in computer science from Brown University.

Article Interest Quotient (Circle One)
High 473 Medium 474 Low 475

The fastest way through customs.

THE PLD DESIGN LANGUAGE WITH SOMETHING TO DECLARE.

You've probably heard about the advantages of programmable logic devices (PLD's) over conventional TTL. In design flexibility, for example. Or increased functional density. Perhaps you've learned the hard way: a competitor using PLD's has beaten you to market with a new product.

What you may *not* have heard, though, is that just switching to PLD's isn't enough. You've got to choose the right PLD design language, too.

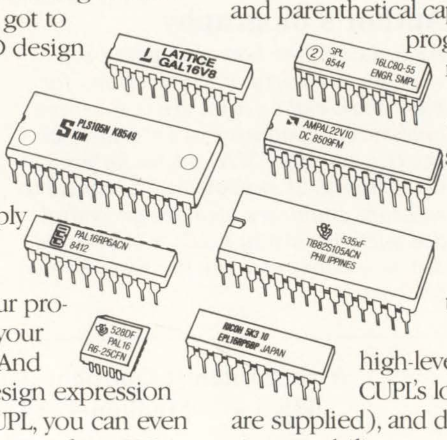
FREEDOM OF CHOICE.

CUPL™ is the one language you can apply to any PLD design. Whatever your PLD device. Whatever your programmer. Whatever your design workstation. And whatever form of design expression you choose. With CUPL, you can even write a logic specification for a PLD before you decide on the target device.

And that means simplified training. Common data structures and design rules. Consistent documentation and testing. Instant adaptability to new devices. The productivity gains with CUPL just keep mounting. In the design cycle. And in production.

POWER TOOL.

CUPL is the most powerful high-level design language for PLD's. It incorporates features like macro substitution, free-form comments, indexed variables and parenthetical capability, to speed program development.



CUPL provides superior state machine capability, and supports all popular models of logic description, including truth tables and high-level equations.

CUPL's logic minimizers (3 are supplied), and deMorgan expansion capability are unequalled.

And CUPL's documentation output file gives you invaluable feedback on your design's progress. These features, plus continual enhancements, make CUPL the benchmark in PLD design languages—flexible and powerful enough for the most sophisticated logic designer's needs.

Now available from Logical Devices, the new version CUPL 4.0 for MS-DOS, VAX-VMS, UNIX C, SUN, APOLLO, and, soon to be announced, for the Apple Mac II Systems.

Also available from Logical Devices, a full line of PROM/PLD programmers.

CUPL.™

LOGICAL DEVICES, INC.

1201 NW 65th Place
Fort Lauderdale, Florida 33309

Toll Free: 800-331-7766
In Florida: (305) 974-0967
Telex: 383142
Fax: (305) 974-8531

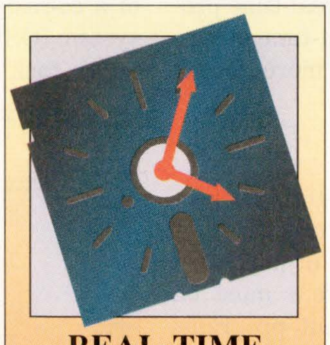
CIRCLE NO. 16

The nature of real time

As real-time programming has matured, a body of knowledge has developed about the special problems of real-time devices and the techniques required to handle these problems. This series of articles will present some of that knowledge in the context of a specific operating system (MTOS-UX) and will demonstrate how to write good, robust real-time programs. Part 1 introduces some basic concepts including real-time programming, multitasking, task attributes, operating systems, and operating-system services.

David L Ripps, *Industrial Programming Inc*

This series is about real-time programming—about programs that run telephone switches, control robot arms, or pilot airplanes. It is not about programs that generate last month's telephone bill, prepare parts lists for items made by a robot, or do cost accounting for an airline. Before you plunge into the technical issues, you should understand exactly why the first is a group of real-time programs and the second is not.



A fundamental property of a real-time program is that some or all of its input arrives from the outside world *asynchronously* with respect to any work that the program is already doing. The program must be able to interrupt its current activity immediately and then execute some predefined code to capture or respond to that input, which is often a fleeting, transient signal. The capture of new data, in turn, may trigger the running of one or more other urgent programs that were waiting for input. Finally, the computer is able to resume its original activity.

Consider a telephone switch that is capable of servicing a rotary dial. Dialing, say, a 5 produces five narrow pulses that must be detected immediately as they fly by. Meanwhile, other parts of the dial processing program must cancel the transaction if the caller hangs up; must time-out the call if there is too long an interval between digits; must start finding a special path if the leading digit is a 1, or an area code, or 800, or 911; and so on. Simultaneously, still other parts of the overall switch program are working on calls that have already been placed: searching for a connection path, generating ring or busy signals, recording billing information, and monitoring hangups, among other activities. This is real time because the timing of the input is completely imposed on the program by unpredictable outside agents (you and me).

In contrast, generating a telephone bill does not have that kind of frantic timing constraint. When the program needs input, it makes a disk access and then waits for the data. (In a multiuser system the billing program may be swapped out during the wait, but that is irrelevant.) If the program takes a long time to compute a bill, there's no catastrophe; the next customer's data will always be there when the program gets around to reading it.

In these two cases the strong difference between real-time and nonreal-time programs arises from the nature of the program specifications—from what the programs have to do. Sometimes the difference arises just from the relative time scales involved. For example, consider a program to control the speed of a rotating shaft. The shaft has a wheel on which are engraved a series of lines that reflect light onto a detector. Shaft speed is computed by counting the number of blips coming from the detector per unit of time or, equivalently, by measuring the

From the book, An Implementation Guide to Real-time Programming, by David L Ripps, ©1989. Excerpted by permission of Prentice-Hall Inc, Englewood Cliffs, NJ.

time interval between blips. (Assume that there is a high-resolution clock available to the program.) The program enters the raw speed measurements into an averaging filter, compares the result with the desired speed, and then computes the power to be sent to the drive motor.

Real-time applications must be written as a series of separate component programs that can execute concurrently.

On the one hand, if the shaft is rotating very quickly or there are many lines on the wheel, the blips arrive faster than the time needed to compute the motor power. The power calculation based on the last period's blip count would have to be interrupted now and then to count the blips for the next period. This is clearly real time. On the other hand, if the speed is low and there are few lines, the program might easily complete the calculation of the motor power in the interval between blips, and then wait to input the next blip. Now, the program has lost its real-time flavor. Thus, the same program requirements can be either real time or not depending upon the time scale of the input in relation to all the work that must be done by the system.

Real-time programs are organized to survive in the face of disturbances that would upset nonreal-time programs. Thus, if there is doubt as to the real-time nature of an application, it is safer to use the more robust real-time organization.

Organization of a real-time program

The need to interrupt one part of a real-time application spontaneously to perform data capture or other more urgent functions forces a particular organization upon such programs. Real-time applications must be written as a series of separate component programs that can execute concurrently. The components are called *tasks* or *processes*; the organization is called *multitasking* or *multiprocessing*. This series will use the terms tasks and multitasking. (The adjective "multiprocessing" is too easily confused with "multiprocessor," which describes a computer system having more than one CPU.)

Each task is a complete program that is capable of independent execution. Each task has a segment of code that it executes. Each task has its own private stack and its own local data areas. These are dedicated memory segments in which the task can keep procedure call parameters, return addresses, temporary data, and similar variables that are not shared with other tasks. Furthermore, each task has its own set of values for the program counter and stack pointer, plus any other general, special, and coprocessor registers that the hardware provides. The hardware register set is known as the execution context, or simply the *context*. (Some tasks may share some or all of their code segment with other tasks. Normally, every task will have the same value for certain special registers, such as those that determine the response to external interrupts. Neither type of overlap lessens the completeness of each task as an executable program.)

Because each task is itself a viable program, each can be started, suspended, resumed, and terminated separately. When an interrupt requires the current task, **C**, to suspend execution in deference to another task, **I, C** stops running, its context is saved, **I**'s context is installed in the hardware, and **I** starts running. Later, **C**'s context is reinstalled in the hardware, and **C** resumes as though there had been no break in its execution (**Fig 1**).

Often, a real-time application has several different kinds of spontaneous input. Some kinds are more important than others; some engender more important responses than others. As a result, task **I** may itself be preempted by yet another task, **I2**.

The real-time operating system

The discontinuous execution of a task is invisible to the task. Real-time programs operate under the control of a co-resident piece of code known as the operating system (OS). The OS is the master program; it decides which task executes on a processor and performs the

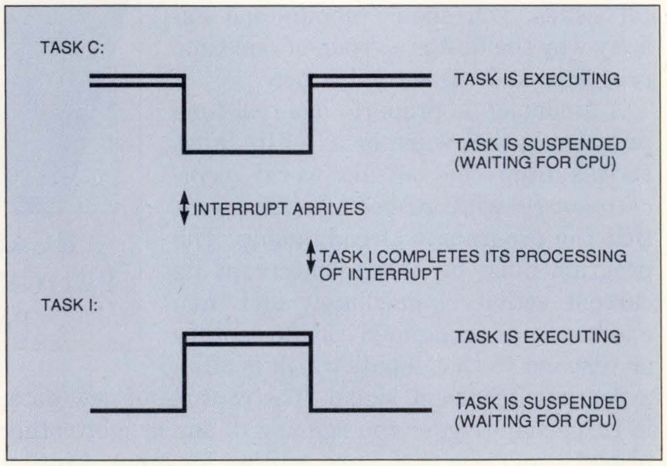

TASK C:
TASK IS EXECUTING
TASK IS SUSPENDED (WAITING FOR CPU)
INTERRUPT ARRIVES
TASK I COMPLETES ITS PROCESSING OF INTERRUPT
TASK I:
TASK IS EXECUTING
TASK IS SUSPENDED (WAITING FOR CPU)

Fig 1—Individual tasks can be started, resumed, and terminated separately. In this example, an interrupt causes task C to be suspended and task I to run. Later, when I must wait for another interrupt, it is suspended, and C resumes running.

Real-time operating systems, revisited

One premise of this series is that real-time services are to be supplied by an operating system or kernel. An operating system is often depicted as layers or concentric circles around the hardware. The innermost layer (the one closest to the hardware) is called the kernel. Thus, in this scheme, a file system is part of the OS proper. The peripheral I/O services that the file system calls are part of the kernel.

Furthermore, we assume that the OS is separate from the application code and is essentially independent of the application. More exactly, the real-time OS can be applied to a wide variety of applications. Alternate approaches are possible.

In the early history of real-time programming, the operating system was not always cleanly separated from the application. A typical real-time program consisted of a series of procedures that performed some function of the application. One procedure might input a certain kind of data by polling at a fixed rate. Another procedure might perform a calculation on the data after polling to determine that the data were available. Tying all this together was a "main program." It served as a crude OS; it called each functional procedure in a preset order. Generally, the main program would permit each procedure to complete before calling the next one.

This cyclic-scheduler approach to real-time programming has fallen into disfavor because it had many problems. First, the cycle times had to be determined by elaborate experiment and tuning. Every time one of the functions was revised, the dynamics of the program changed and the system had to be retuned. As a result, the programs were a nightmare to maintain and expand.

Equally annoying was the fact that the ad hoc approach to scheduling and service delivery required a new main program for each application. There was too much application-specific code inside the "operating system." Furthermore, there was little hope of making the execution efficient by performing actions in parallel. Instead of interrupts and preemption there was polling and serial execution. Thus, when a message had to be output to a console, the entire program waited while each character went out.

There is yet another approach to real-time programming that is not so easily dismissed. Time may even show that it is the right way to proceed. The idea is to build real-time facilities directly into the task language. Concurrent Pascal, Modula, and Ada incorporate this technique.

The chief advantages of having real-time features within the tasking language are increased portability and possibly enhanced error checking. In principle, the compiler can do extensive checking for misuse of real-time facilities and potentially dangerous practices.

The chief disadvantage of the linguistic approach is that the real-time primitives may not be adequate or appropriate for your application. Ada provides only one mechanism for coordination and communication among tasks—the rendezvous. But the rendezvous has some very unfortunate semantic properties that were not fully appreciated until after the language had been frozen. For example, the First International Workshop on Real-Time Ada Issues reported that a low-priority task can delay a high-priority task for an unlimited amount of time because the rendezvous has first-come, first-served queuing. These problems are likely to be corrected in the next version of Ada—Ada-9X—which won't be introduced until well into the 1990s.

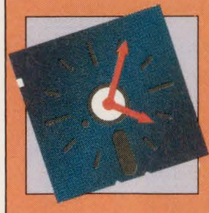
Ada restricted the number of real-time features to minimize other problems: compiler size and complexity. The more features you have in a language, the longer the compilation takes. At present, relying upon an independent OS for real-time facilities seems to be a good practical approach to real-time programming.

This series presents each aspect of real-time programming purely from the application (task) side; it does not explain how to write the operating system that supplies the required services. Robust real-time operating systems are commercially available. It is far more economical in money and project time to buy an OS than to plan, write, debug, test, and maintain one of your own. There is no more reason to develop a private OS than to develop a private editor, compiler, assembler, or linker. If you still doubt that point, see "No Silver Bullets—Essence and Accidents of Software Engineering," in particular, the section headed "Buy vs Build" (**Ref 1**).

The real-time OS selected for this series is MTOS-UX, a product of Industrial Programming Inc (Jericho, NY). MTOS-UX is available for several families of processors, such as the Motorola 680xx and 88x00, Intel 80x86, and National Semiconductor 32x32.

required context switches. It also handles the hardware interrupts that normally announce the availability of fresh input and determines when the response task is to be activated. In short, the OS schedules all processor work.

Thus, a real-time program consists of a set of tasks—separate programs that compete for access to the CPU—and the OS—a master program that schedules CPU access. The work done by the OS in scheduling and context switching is pure overhead; it decreases the time available for task work. Nevertheless, this small loss is easily repaid. The OS makes sure that the CPU is kept busy as long as there is any task work to do.



The operating system decides which task executes on a processor and performs the required context switches.

Consider the alternatives. Suppose that task **I** were executing and reached a point at which it had to have input to continue. If there were no OS, **I** would have to sit in a polling loop, repeating the question: Has the input arrived yet? This is a terrible waste of CPU time. Instead, the OS gives the CPU to task **C**. Task **C** can accomplish productive work until the interrupt arrives. At that point (with slight additional overhead), the CPU can be turned over to **I** to process the input.

Because of its role as central authority over all tasks and interrupts, the OS is also in the best position to provide centralized services and to control access to hardware and software facilities that are shared by the tasks. Thus, timekeeping, peripheral I/O, and allocation of memory are all in the domain of the OS.

Handling such chores as peripheral I/O within the OS yields considerable space efficiency over duplicating the code in each task that needs the function. Furthermore, the individual tasks rarely have the application-wide information required to resolve the conflicts that arise when allocating shared resources. Centralizing also shortens development time for real-time applications. A debugged general OS can usually be applied immediately to a new project.

In real-time work it is common to have dozens of concurrent tasks. Some have been suspended by the arrival of input that merited immediate attention. Others have been blocked because they requested a service of the OS and that service is not yet complete. (Incomplete services might involve a requested disk access that is still in progress, or some requested memory that cannot yet be allocated.)

Task states

To help track the activities of various tasks, OSs commonly maintain the state of each task. One scheme is to classify a task as Running if it is currently executing on a processor, as Ready if it can execute as soon as a processor becomes available, as Blocked if a requested service is not yet complete or there is some other impediment that prevents its execution, and as Dormant if it has terminated after it finished executing or it has not yet been requested to execute. This is the scheme used in this series of articles. The possible

Language issues

Although this series concentrates on the concepts of real-time programming, these concepts can become fuzzy until you see specific examples coded in real language for a real operating system. The MTOS-UX was chosen as the OS. The language chosen was C, based mainly on its popularity among real-time programmers and its ease of use. Furthermore, if one resists the urge to be overly terse, C can be as clear as any other high-level language. Those who are unfamiliar with C might want to read at least the early chapters of Kernighan and Ritchie **(Ref 2).**

Tasks need not be written in C; all MTOS-UX services can be invoked from assembly language as well as from Fortran, Pascal, or other high-level languages. As Part 2 of this series will discuss, a programmer requests an OS service by calling a procedure, such as *pause* (10+MS) to pause for 10 msec. All high-level languages that are likely to be used in real-time applications provide some mechanism to call procedures with arguments. In fact, many compilers have the option of generating calls that are compatible with C. Thus, as far as the OS services are concerned, it doesn't matter if the program is expressed in C or in any other language with a compatible interface.

Nevertheless, C, Fortran, and Pascal do share a property that sets them aside from a new group of languages typified by Ada. Ada has tasking and other real-time facilities built directly into the language; C, Fortran, and Pascal do not.

transitions between these four states of a task are shown in **Fig 2**. (It is not necessary to make such a strong distinction between Ready, Dormant, and Blocked states. A Ready task can be considered blocked waiting for a CPU to become available. Similarly, a Dormant task is just waiting to be restarted. In this view, every task is either Running or is Blocked, waiting for something.)

Different operating systems employ different strategies to choose which task to run next when there is more than one Ready task. The simplest rules are first-come, first-served and equal-execution-time-slice-for-all-tasks. However, at a given moment within a real-time application, each activity has a discernible level of importance or priority with respect to other ongoing activities. Consequently, the tasks that perform the activities carry a corresponding level of priority. A good real-time OS maintains not only the state of each task but the current priority as well. If there is more than one Ready task, the highest priority (most urgent) one gets the processor. Among tasks of equal priority, it can be first-come, first-served.

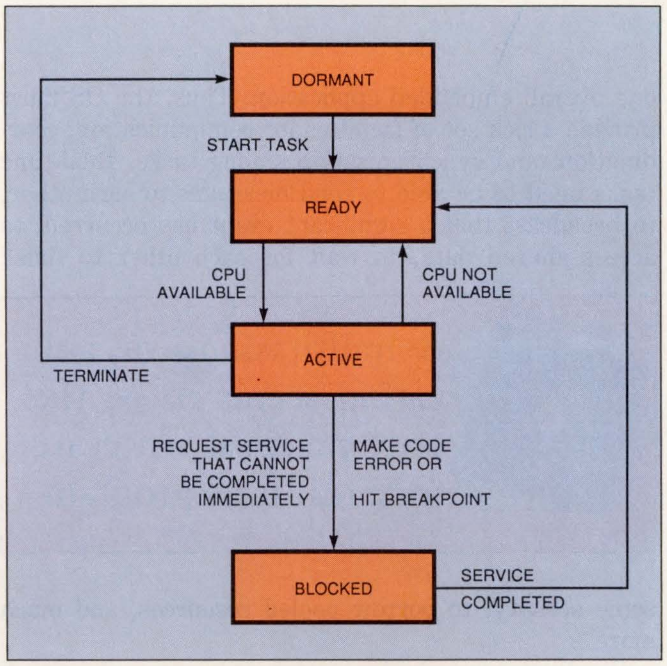


Fig 2—An operating system maintains the state of each task. A task is Running if it is currently executing on a processor; Ready if it can execute as soon as a processor becomes available; Blocked if a requested service is not yet complete or there is some other impediment that prevents its execution; and Dormant if it has terminated after it finished executing or it has not yet been requested to execute.

Further criteria for a real-time program

A multitasking organization seems to be the only practical structure for a real-time program. However, programs that have neither spontaneous data nor other aspects of real time to cope with may also employ multitasking. For example, Unix, VAX VMS, and OS/370 all use multitasking, even though these operating systems were not designed to support real-time work. Besides the presence of asynchronous input, two further properties seem to complete the definition of a real-time application: a high degree of interdependence among the component tasks and a need to have very rapid response to interrupts.

In a nonreal-time environment, there is little or no relation among the tasks that are running concurrently; they are a collection of independent programs that happen to have been submitted by the various independent users of the hardware. Thus, the OS need not and does not provide strong facilities to coordinate the tasks. Quite the contrary, for security reasons, the OS tries to make it difficult for one task to interfere with another.

In contrast, in a typical real-time program all of the tasks are highly interrelated: They are all aspects of

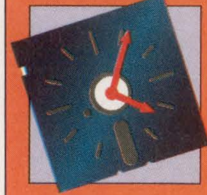
Companion disk offer

All of the C examples in this series, plus applications of your own, can be run on a personal computer with a set of demonstration disks available from Industrial Programming Inc. The disks contain a full version of MTOS-UX for an IBM PC/AT or compatible. An application program is edited, compiled, linked, and loaded under MS-DOS. The MTOS-UX then takes over the hardware to execute the program in real time. At any time, you can enter an alt/dlt command from the console to return control to MS-DOS.

The demonstrator requires an IBM PC/AT with at least 512k bytes of RAM and a hard disk with 2M bytes available for MTOS libraries and scratch storage. Program preparation requires the Microsoft C compiler/linker, version 5.0 or later. Microsoft tools are not included with the MTOS-UX demonstrator.

The demonstration version has all of the features and facilities of standard MTOS-UX. However, there is a limit of six of each type (six tasks, six mailboxes, six semaphores, and so forth). The disk set costs $25; unlimited versions are also available. For more details, call the IPI sales department at (800) 365-6867.

one overall embedded application. Thus, the OS must provide a rich set of facilities for communication, coordination, and synchronization among tasks. Real-time tasks need to be able to send messages to each other, to broadcast that a significant event has occurred, to access shared data, to wait for each other to finish



A multitasking organization seems to be the only practical structure for a real-time program.

some activity, to borrow pooled resources, and much more.

Finally, very rapid response to external stimuli seems to be inherent in the real-time world. The maximum time to recognize that an external interrupt is pending (latency) plus the time needed to suspend the current task and switch to an interrupt handler (context switch time) is commonly on the order of microseconds. Unix, and similar nonreal-time operating systems, were not designed to react so quickly.

This series will continue in regular issues of EDN, beginning with the October 1st issue. Part 2 will further explore the nature of real-time programming and describe how the operating system handles the special concerns imposed by working in real time. **EDN**

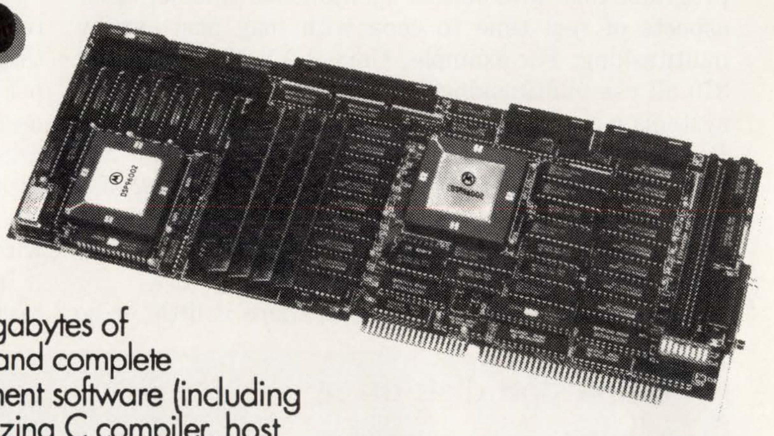
References

1. Brooks, Jr, Frederick P, "No Silver Bullets—Essence and Accidents of Software Engineering," *Unix Review*, November 1987.

2. Kernighan, B and D Ritchie, *The C Programming Language*, Prentice Hall, Englewood Cliffs, NJ, 1978.

Article Interest Quotient (Circle One)
High 494 Medium 495 Low 496

100 MFLOP ENGINE



Based on Motorola's 50 MFLOP 96002, Ariel's Dual DSP MM-96 blasts through real time signal processing, graphics, floating point number crunching and multimedia applications like nothing else.

The MM-96 hooks directly to frame grabber cards via its DT-Connect™ interface and to digital audio with Ariel's DSPnet™ multimaster bus.

Configurations for IBM AT compatibles are available with up to 16 megabytes of memory and complete development software (including an optimizing C compiler, host drivers, and demo software).

Ariel provides the best applications support in the business via telephone, mail, fax, or our 24 hour DSP BBS.

The MM-96 is available now. Call for 96002 support on other platforms.

Ariel_

Ariel Corporation
433 River Road
Highland Park, NJ 08904
Telephone: (201) 249-2900
Fax: (201) 249-2123
DSP BBS: (201) 249-2124

DT-Connect is a trademark of Data Translation, Inc. DSPnet is a trademark of Ariel Corporation.

c 1990 Ariel Corporation.

Solutions for today are created by those with a vision of tomorrow.

Advanced real-time systems have special needs. In today's world, system software must be designed for tomorrow's computers. Software designers know that performance and ease of development are critical for real-time operating systems. PDOS–your performance real-time operating system–has been dramatically improved. Our new 4.0 release has incorporated those features critical to the future of complex system designs. PDOS 4.0 makes your development time cost-effective and productive. Also included is support for one full year at no additional cost. We offer superb training on our products to keep learning curves to a minimum. Today, you can have a vision of tomorrow by developing with the performance real-time operating system–PDOS.

© 1990 Eyring

1455 West 820 North
Provo, Utah 84601
Tel: 801-375-2434
Fax: 801-374-8339

PDOS®

EYRING
Systems Software Division

Benelux
Interay BV
Lageweg 2A
9251 GM Bergum
The Netherlands
✆ 05116 / 4052
Fax 05116 / 2698

Germany, Switzerland, Austria
Systrix GmbH
Hindenburgring 31
D-7900 Ulm/Donau
West Germany
✆ 0731 / 37515
Fax 0731 / 37510

United Kingdom
Eyrisoft Ltd.
Etwall Street
Derby DE3 3DT
England
✆ 0332 / 384978
Fax 0332 / 360922

Israel
Militram
P.O. Box 13324
Ramat Hachayal
Tel Aviv 61330
Israel
✆ 97252545685
Fax 97252574383

Japan
Hitachi Zosen Corporation
Hi-System Division
3-4, Sakurajima 1-Chome
Konohana-ku, Osaka 554
Japan
✆ (06) 465-3172
Fax (06) 465-4045

CIRCLE NO. 9

Bye-bye

Now there's a way to solve your software development backlog problems.

And Digital has it today.

It's the only kind of solution to the problems of developing software that really works.

A total solution.

It's Digital's complete CASE environment. It gives developers of commercial and technical applications a totally integrated approach to software development – something that's essential to the software development cycle and accelerates it in ways that CASE tools alone never could.

❑ WRITE ONCE AND FOR ALL.

What's so unique about Digital's CASE environment is what it lets you do. That's because it rests solidly on a foundation of architectural standards that are both open and flexible.

A case in point. Our CASE tools are supported by Digital's Network Application Support (NAS). Digital's NAS lets you develop applications for computers with one operating system, yet run them on different computers with different operating systems. The competition can't offer this level of integration for saving time and money.

❑ A FRAMEWORK THAT REALLY WORKS.

We also offer a CASE integration framework, specifically designed for software development. As with our architectural standards, the framework is open, flexible and complete.

digital™

backlog.

For example, it offers the industry's first distributed CASE repository. Consequently, team communications, process management, data and information sharing and other functions that make development faster and easier are integrated. What's more, 3rd party products and tools can also be integrated.

❑ THE COMPLETE TOOL CASE.

Then there are the tools themselves. Here too, Digital offers more. We provide a complete set of industry-leading tools for every aspect of the development cycle. These include tools for information systems, transaction processing, technical, scientific and embedded applications and more.

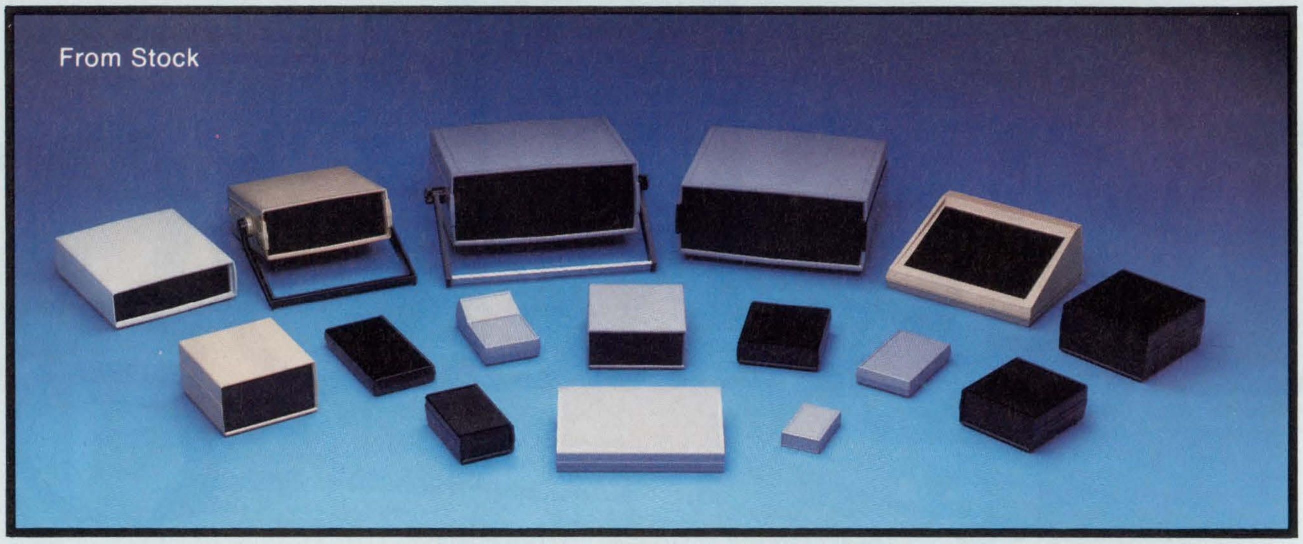
❑ A SUPPORTIVE ENVIRONMENT.

And finally, there's support. As with everything else we offer for our CASE environment, our support is all-encompassing. Count on things like training, consulting, special courses, worldwide service and even CASE integration services.

To be effective and productive today, you need the right tools. But, more important than that, you need the right environment. Find out now what a difference Digital's complete CASE environment can make. Call 1-800-842-5273 ext. 315. Or call your local Digital sales office.

Digital
has
it
now.

Electronic Enclosures...
from stock - or - custom modified



Stock Enclosures • Modifications • Design Engineering Assistance • Custom Panels • Options and Accessories

- Easy to design into and easy to assemble
- Molded-through color means no chipping or scratching—and no need for refurbishing and painting during or after assembly
- Constructed of impact-resistant ABS—(flame-retardant grade to meet UL94V-0 standards, optional)
- Shielding against EMI/RFI available
- Standard colors: tan, gray, black, PC bone
- No tooling costs or set-up charges
- Molded-in mounting bosses, card guides, and panel grooves reduce assembly time and production costs
- Low-cost options and accessories available to meet end-user needs
- Available in kits and production quantities

Pac-Tec enclosures from stock, or modified "Your Way" by Pac-Tec's unique method of tool modification are available from your local stocking Distributors. For the name of your local distributor or additional information call:

PAC-TEC®
Division of LaFrance Corp.
Enterprise and Executive Avenues Philadelphia, PA 19153 Telephone (215) 365-8400 Telex 50-6082 FAX: 215/365-4420

1-800-523-4813

Put LabWindows to Your Test

LabWindows® INSTRUMENTATION SOFTWARE

A+

Data Acquisition ⟷ **Data Analysis** ⟷ **Data Presentation**

Data Acquisition	Data Analysis	Data Presentation
GPIB	Digital Signal Processing	File I/O
Plug-in Boards	Statistics	2D Plots
VXI	Curve Fitting	Real Time Strip Charts
RS-232	Array Operations	Printer and Plotter Output

True False

☑ ☐ Industry standard Microsoft C, QuickC, and QuickBASIC development tools for data acquisition and instrument control.

☑ ☐ Instrument drivers for over 100 GPIB, VXI, and RS-232 instruments.

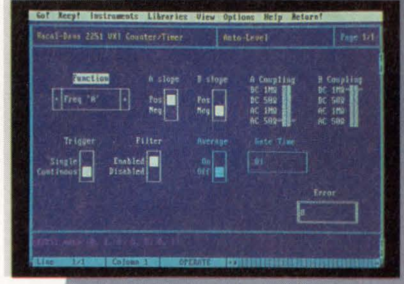
☑ ☐ Integrated support for plug-in data acquisition boards.

☑ ☐ Powerful analysis library for real-time data processing.

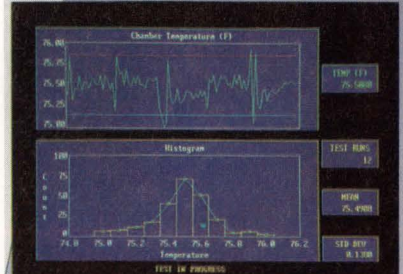
☑ ☐ Extensive graphics library for creating full-color displays on printers and plotters.

☑ ☐ Code generation and debugging tools for faster program development.

☑ ☐ Software development tools for production test and ATE systems.



Go! Keep! Instruments Libraries View Options Help Return!
Racal-Dana 2251 VXI Counter/Timer
Auto-Level
Function
A slope
B slope
A Coupling
B Coupling
Trigger
Filter
Average
Gate Time
Error



LabWindows...making industry standard programming languages work for you in data acquisition and instrument control.

National Instruments Italy (02) 4830 1892
National Instruments France (1) 48 65 33 70
National Instruments Switzerland (056) 82 18 27
National Instruments United Kingdom (06) 35 523 545
Nihon National Instruments K.K. (Japan) (03) 788 1921

NATIONAL INSTRUMENTS®
The Software is the Instrument®
6504 Bridge Point Parkway
Austin, TX 78730-5039

Call for a *FREE* Demo Disk and Catalog
(512) 794-0100 • (800) 433-3488 (U.S. and Canada)

CIRCLE NO. 12

SOURCEGATE

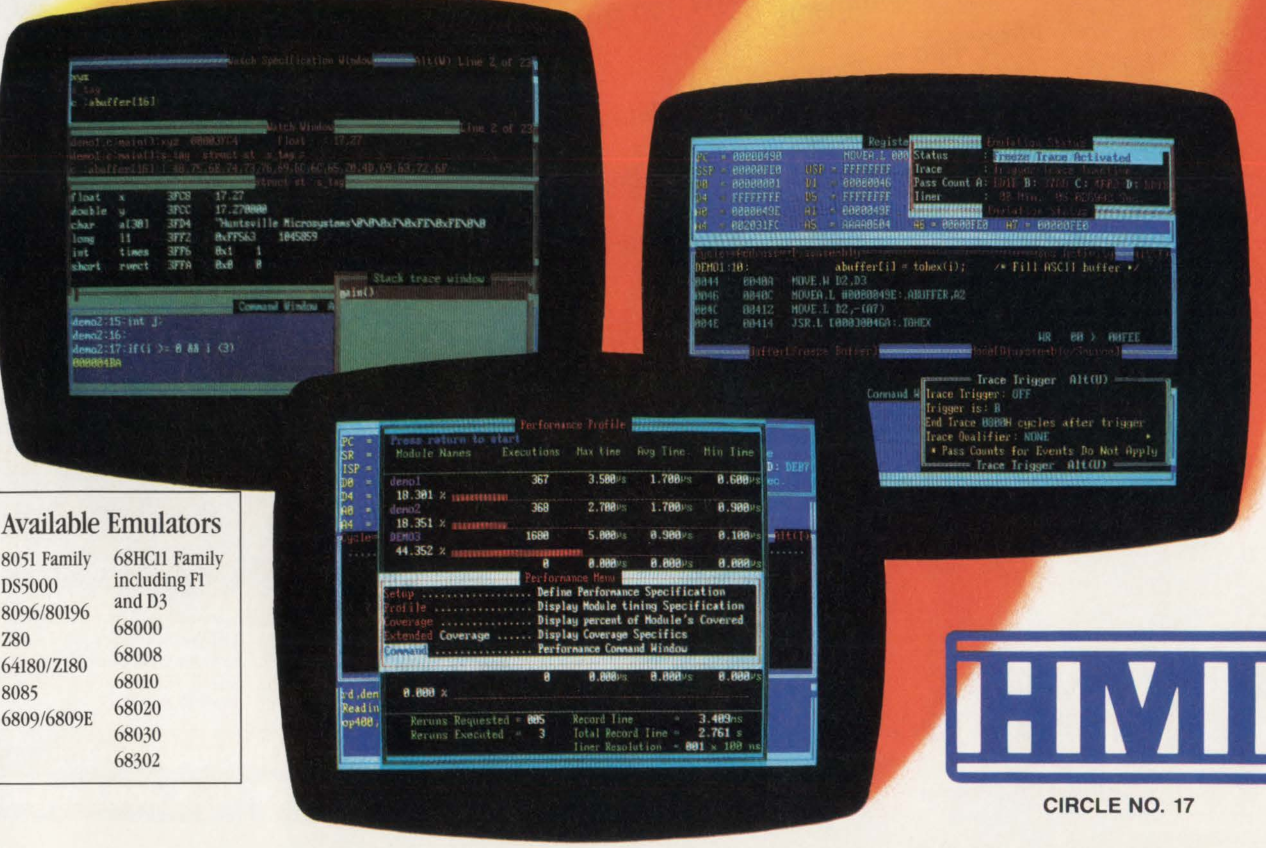
integrating high level language debugging with in-circuit emulators.

SourceGate is a window driven high level language debugger designed to support the Huntsville Microsystems 200 series of in-circuit emulators.

- User configurable windows can be sized, moved and duplicated anywhere on the screen.
- Code can be viewed in all displays (trace, single step, etc.) in one of three modes: Source only, Assembly only or both Source and Assembly.
- Watch windows display and monitor code variables.
- Optional Performance Analysis Card for real-time software performance analysis and real-time software test coverage.
- Available for IBM PC family and UNIX systems including Apollo and SUN.

For more complete technical information, write to Huntsville Microsystems Inc., 4040 South Memorial Parkway, Huntsville, AL 35802 or call (205) 881-6005.

IBM is reg. T.M.
International Business Machines, Inc.
Unix is reg. T.M.,
Bell Laboratories, Inc.



```
Watch Specification Window      Alt(W) Line 2 of 23
xyz
s_tag
c :abuffer[16]
                Watch Window                Line 2 of 23
demo1.c:main():xyz  00003FC4      float   : 17.27
demo1.c:main():s_tag  struct st  s_tag :
c :abuffer[16] : 30,75,6E,74,73,76,69,6C,6C,65,20,4D,69,63,72,6F
                struct st  s_tag
float   x      3FC8   17.27
double  y      3FCC   17.270000
char    a[30]  3FD4   "Huntsville Microsystems\0\0\0xF\0xFE\0xFE\0\0
long    l1     3FF2   0xFF563   1045859
int     times  3FF6   0x1    1
short   rvect  3FFA   0x0    0
                                        Stack trace window
                                        main()
                 Command Window
demo2:15:int j;
demo2:16:
demo2:17:if(i >= 0 && i <3)
000004BA
```

```
                  Registe          Emulation Status
PC  = 00000490         MOVEA.L 000 Status      : Freeze Trace Activated
SSP = 00000FE0  USP = FFFFFFFF     Trace       : Trigger Trace Inactive
D0  = 00000001  D1  = 00000046     Pass Count A: [illegible] B: [illegible] C: [illegible] D: [illegible]
D4  = FFFFFFFF  D5  = FFFFFFFF     Timer       : 00 Min. 05.025945 Sec.
A0  = 0000049E  A1  = 0000049F            Emulation Status
A4  = 002031FC  A5  = AAAA0604    A6 = 00000FE0   A7 = 00000FE0
Cycle=Addressed[Disassembly]
DEMO1:10:              abuffer[i] = tohex(i);     /* Fill ASCII buffer */
0044    0040A   MOVE.W D2,D3
0046    0040C   MOVEA.L #0000049E:.ABUFFER,A2
004C    00412   MOVE.L D2,-(A7)
004E    00414   JSR.L [000]0046A:.TOHEX
                                              WR   00 >  ABFEE
     Buffer[Freeze Buffer]          Mode[Disassembly/Source]

                   Trace Trigger  Alt(U)
Command W Trace Trigger: OFF
          Trigger is: B
          End Trace 0000H cycles after trigger
          Trace Qualifier: NONE                      *
          * Pass Counts for Events Do Not Apply
                   Trace Trigger  Alt(U)
```

```
                    Performance Profile
PC  =  Press return to start
SR  =   Module Names      Executions   Max time   Avg Time   Min Time
ISP =
D0  =  demo1                  367      3.500μs    1.700μs    0.600μs
D4  =   10.301 %
A0  =  demo2                  368      2.700μs    1.700μs    0.900μs
A4  =   10.351 %
Cycle= DEMO3                 1680      5.000μs    0.900μs    0.100μs
        44.352 %
                                0      0.000μs    0.000μs    0.000μs
                    Performance Menu
Setup .................... Define Performance Specification
Profile .................. Display Module timing Specification
Coverage ................. Display percent of Module's Covered
Extended Coverage ........ Display Coverage Specifics
Command .................. Performance Command Window
                                0      0.000μs    0.000μs    0.000μs
rd,den  0.000 %
Readin
op400,   Reruns Requested = 005    Record Time        =  3.403ns
         Reruns Executed  =   3    Total Record Time =   2.761 s
                                   Timer Resolution  = 001 x 100 ns
```

Available Emulators

8051 Family	68HC11 Family including F1 and D3
DS5000	68000
8096/80196	68008
Z80	68010
64180/Z180	68020
8085	68030
6809/6809E	68302

HMI

CIRCLE NO. 17

Mix C and assembly language for fast real-time control

Flow-control code compiled in C often executes too slowly for real-time applications. But if you replace switch statements with an assembly-language driver, your program will run almost twice as fast as a pure C program.

Rick Brown, *Desert Research Institute, University of Nevada*

Programming in C gives you close control over hardware, and code that's easy to write, debug, and understand, but its compiler-generated code for program-flow control runs too slowly to suit high-speed real-time or control applications. Also, C provides no standard interrupt-processing capabilities. Assembly language, on the other hand, generates code that both executes faster than compiled C and can process interrupts, but the code is tedious to write and difficult to understand.

A method of combining the two languages preserves the clarity of the C language while minimizing its time penalties. Using C language makes designing the system logic easy; including an assembly-language driver, which controls a chain of small C functions, speeds up the transfer of control from one C function to another. This combination technique can speed up a system by 25 to 100%. When considering potential uses for an assembly-language driver, your first question will naturally be "What is the code really doing?" You'll find that, essentially, the code substitutes a simple address calculation for a C switch statement and its multiple tests to find a specific case.

A controller in an industrial laundry-folding machine illustrates this programming method. The folding-machine controller requires servicing of software routines at precise time intervals as well as moderately high-speed operation.

The folding machine processes material at a rate of 150 ft per minute. This rate requires that sensors and actuators be serviced about every five milliseconds in order to achieve the required ¼-in. resolution. The machine optically detects the leading and trailing edges of sheets of material to be folded, and it times the control functions from these edges. The five folding stations in the machine allow as many as five pieces of material to be in the machine at any one time. Thus, only about one millisecond out of five can be budgeted to track the progress of a single piece of material through the machine.

For the machine to use a Z80180 microprocessor running at 4.6 MHz, a control program written entirely in C is not fast enough. The time required for processing a C switch statement is too long. For example, a 7-case switch requires 175 to 257 μsec, depending on whether it detects the first or last case. In this case, one control statement uses more than 25% of the time budget. Assembly-language code, some of which is necessary anyway for interrupt processing in the controller, can provide a faster method of selecting a function, and thus speed up the system.

An interrupt-driven assembly-language driver can link together C functions and minimize control overhead in real-time applications.

The combination programming method uses numerous small C functions, which a segment of the assembly-language portion of the code (the "function driver") activates. Tracking a "control sequence" (the events and actions that constitute the processing of a single piece of material through the machine) requires about 20 C functions. This article refers only to the portions of the application code activated at 5-msec intervals by an interrupt request from the Z80180's counter timer. (Operator communication and other tasks that are not time critical are controlled by function-pointer stacks (**Ref 1**).)

Each C function controls a single part of the folding process. At each 5-msec timer interrupt, the function driver activates one or more of the C functions. For example, the first function continually checks to see if a new piece of material is entering the machine. As soon as a piece enters, the system activates a second (control) function, which waits for a predetermined time and then closes the first actuator. The function driver then transfers control to a third function, which looks for the tail end of the material at the machine entrance. Control continues to pass from function to function until the piece of material exits the machine.

Control flow is determined by the values that the C functions return. The return value is an index into a table that contains pointers to all of the possible C functions. Each time a function is called, it returns the index of the next function in the control sequence if it has completed its operation, or it returns its own index if its operation is incomplete. **Fig 1** is a control-flow diagram for the first five C functions.

In addition to performing some I/O services, the function driver maintains a control array containing the table indices and the working variables returned by the C functions. At each interrupt, the function driver transfers control in turn to each of the functions whose function-pointer-table indices are currently in the control array. After all entries in the control array have been satisfied, the machine idles or performs tasks that are not time critical until the next time interrupt.

Control-array structure

The control array consists of a FIFO circular buffer. Each entry in the buffer contains the index of a function to be activated, a time word, and a temporary storage word that active control sequences can use. The function driver maintains pointers to the bottom (oldest) and top (youngest) locations of the FIFO buffer. The youngest entry in the buffer is always the sensing function that waits for a new piece of material to enter the folding machine. It continually returns its own index until new material arrives; then it returns the index of the first control function. The function driver detects that occurrence, increments the buffer-top pointer, and places a new copy of the index of the "material-entering" function at the new buffer-top location. The function driver also monitors the value returned by the oldest control sequence in the buffer; when the function driver finds a 0 (a "process-complete" flag), it increments the buffer-bottom pointer, thereby removing that control sequence from the buffer.

Code directs function sequence

Listing 1 (see pg 46) presents a sample of the C source code that controls the first five functions of the folding machine. Lines 1 to 14 define function 0 (fa_start), which waits for new material at the entrance of the machine. This function is the first entry in the function-pointer table, so its index is 0.

The parameter *time* is the address of the time word for the current control sequence in the FIFO buffer. The function driver increments the contents of the time word for each control sequence at each timer interrupt. The C language routines may use or alter the contents of the time word as necessary; for example, at line 10 the contents are set to 0. If the machine is stopped, as determined by the nontime-critical portion of the machine code, the fa_start routine exits at line 5.

The function driver maintains the variable FA_PDREG; the contents of this variable are bits that reflect the state of the photosensors that detect the presence of material at each folding station. B1PD is the bit position in FA_PDREG of the photosensor at the machine entrance. The global variable fa_pd1clear lets the system detect the end of one piece of material and the beginning of a new piece. If the previous piece has not moved clear of the machine entrance, the routine exits at line 7. Otherwise, the routine waits for the entry of a new piece and counts it at line 9. The routine exits at line 11 and returns the index of the next routine. If no material is present at the machine entrance, then the routine sets the fa_pd1clear variable to reflect that condition and exits at line 13. No code (such as a timeout) exists to handle the possibility that the head of piece B will overlap the tail of piece A; defensive programming would call for an error-handling routine to cover this condition. In

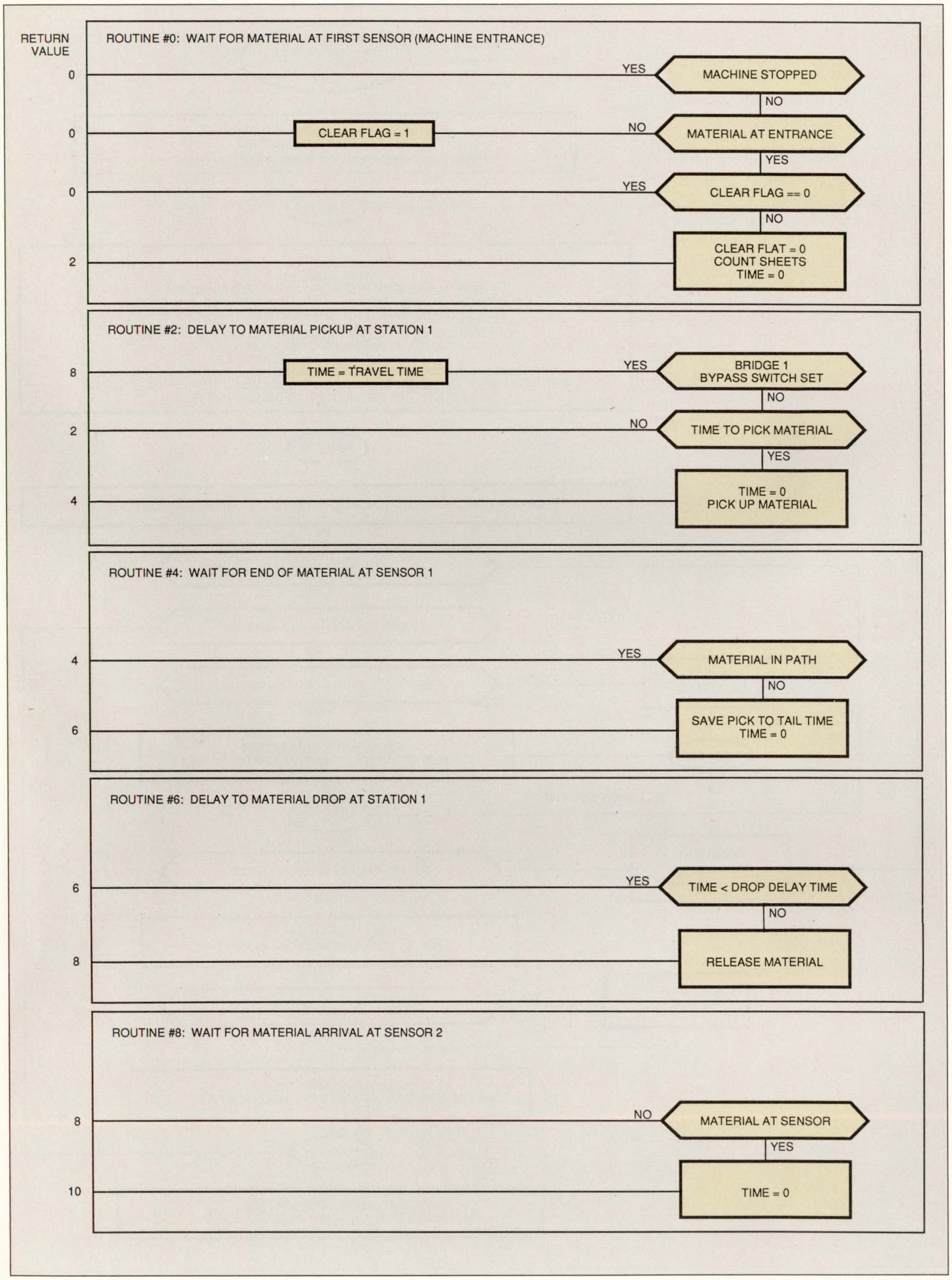


Fig 1—A C function implements each part of the control process. When an operation completes, the function returns the index of the next function in the control sequence.

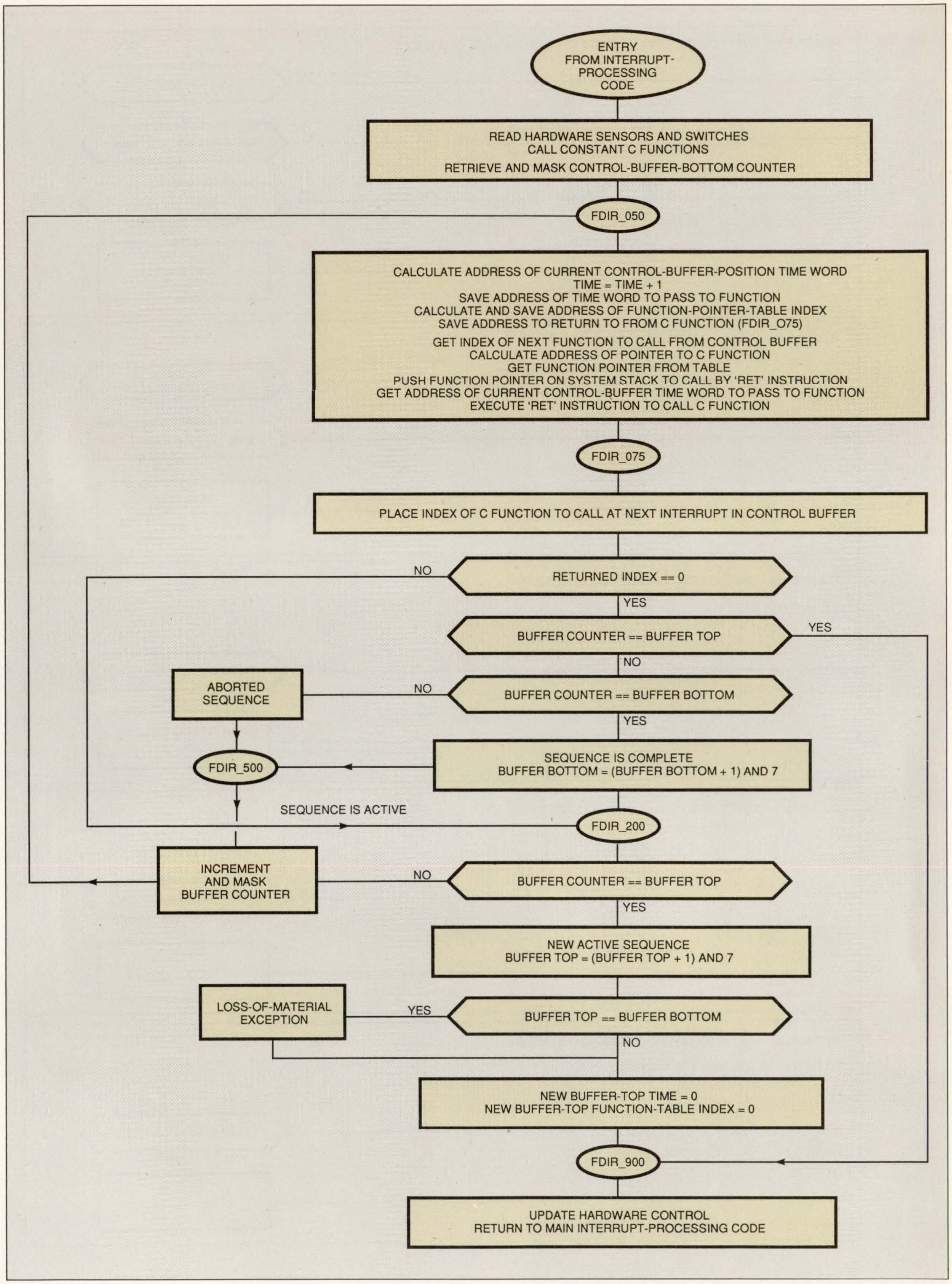

ENTRY FROM INTERRUPT-PROCESSING CODE
READ HARDWARE SENSORS AND SWITCHES
CALL CONSTANT C FUNCTIONS
RETRIEVE AND MASK CONTROL-BUFFER-BOTTOM COUNTER
FDIR_050
CALCULATE ADDRESS OF CURRENT CONTROL-BUFFER-POSITION TIME WORD
TIME = TIME + 1
SAVE ADDRESS OF TIME WORD TO PASS TO FUNCTION
CALCULATE AND SAVE ADDRESS OF FUNCTION-POINTER-TABLE INDEX
SAVE ADDRESS TO RETURN TO FROM C FUNCTION (FDIR_O75)
GET INDEX OF NEXT FUNCTION TO CALL FROM CONTROL BUFFER
CALCULATE ADDRESS OF POINTER TO C FUNCTION
GET FUNCTION POINTER FROM TABLE
PUSH FUNCTION POINTER ON SYSTEM STACK TO CALL BY 'RET' INSTRUCTION
GET ADDRESS OF CURRENT CONTROL-BUFFER TIME WORD TO PASS TO FUNCTION
EXECUTE 'RET' INSTRUCTION TO CALL C FUNCTION
FDIR_075
PLACE INDEX OF C FUNCTION TO CALL AT NEXT INTERRUPT IN CONTROL BUFFER
NO
RETURNED INDEX == 0
YES
BUFFER COUNTER == BUFFER TOP
YES
NO
ABORTED SEQUENCE
NO
BUFFER COUNTER == BUFFER BOTTOM
YES
FDIR_500
SEQUENCE IS COMPLETE
BUFFER BOTTOM = (BUFFER BOTTOM + 1) AND 7
SEQUENCE IS ACTIVE
FDIR_200
INCREMENT AND MASK BUFFER COUNTER
NO
BUFFER COUNTER == BUFFER TOP
YES
NEW ACTIVE SEQUENCE
BUFFER TOP = (BUFFER TOP + 1) AND 7
LOSS-OF-MATERIAL EXCEPTION
YES
BUFFER TOP == BUFFER BOTTOM
NO
NEW BUFFER-TOP TIME = 0
NEW BUFFER-TOP FUNCTION-TABLE INDEX = 0
FDIR_900
UPDATE HARDWARE CONTROL
RETURN TO MAIN INTERRUPT-PROCESSING CODE

Fig 2—The assembly-language function driver makes decisions based on the values returned by each C function. The driver also maintains the table of function pointers and a control buffer.

High-speed applications always need some assembly-language code for time-critical tasks; you can use that code as a means to improve overall system speed.

practice, however, the material enters the machine so fast that it's impossible for the operator to move quickly enough to cause an overlap.

Lines 15 to 26 define function 2 (fa_b1pick), which controls the first actuator. (The routine numbers (indices) increase by two because routine addresses are 2-byte numbers in the function-pointer table.) The function driver maintains the FA_SENSE variable, which contains the state of the operator switches. If the operator presses a control button to bypass the first actuator (for example, because the material is too small to need two folds), then the content of FA_SENSE causes the function to exit immediately at line 21, transferring control to routine 8. Otherwise, the routine monitors time at line 22 until it's time to operate the first actuator; then (at line 25) it transfers control to routine 4, which energizes the actuator.

The variable FA_CTLRG contains a bit position for each actuator. The C functions control the actuators by setting or resetting bit positions in this variable (line 24). The function driver passes the contents of FA_CTLRG to the hardware interface for use in selecting and energizing one or more actuators.

Other points to note in the C code are line 20, where the time word is preset for control of subsequent routines, and line 32, where the routine uses a temporary-storage location in the circular buffer to store an intermediate result. Also, in line 41, routine 6 modifies FA_CTLRG to turn off the actuator that was turned on at line 24. When a piece of material finally exits the machine, the last function executes a *return(0)* statement that terminates the control sequence. Remember that several control sequences may be active at any one time.

Fig 2 is the control-flow diagram of the assembly-language function driver, which receives the calls at every 5-msec timer interrupt and links the C functions into their control chain. **Listing 2** (see pg 47) presents the corresponding Z80 assembly-language source code. At lines 9 through 14, the driver reads the hardware photosensors and operator switches, and saves their states in global variables. Line 15 is a call to a C function that is activated unconditionally at every timer interrupt. Lines 16 through 19 set the buffer counter to the oldest buffer entry.

The loop that processes each buffer entry begins at line 21 and ends at line 125. For each control sequence, lines 21 through 39 calculate the buffer address and increment the time word. Lines 40 through 62 make the address of the control-sequence time word accessible to the C function and also retrieve the address of the pointer to a C function from the function-pointer table. Line 63 transfers control to the C function.

In lines 65 to 82, the driver makes its decisions about the activity of the C function on the basis of the returned index and current buffer counter. It is possible that the processing of a piece of material may be aborted during a control sequence. Should that occur, lines 83 to 85 cause the control sequence to idle until it becomes the oldest (bottom position) in the buffer. Lines 87 to 125 control the buffer-top and -bottom counters and the execution path through the control loop.

In lines 127 through 132, the driver sends the results of control decisions to the hardware. At line 133, the driver returns control to the interrupt-processing system, which can then work on the nontime-critical portion of the code.

```
int  (*fp_table[32])();         /*function pointer table*/
int  func_1(), func_2(), ...;   /*subject functions*/
int  index;
{
 ...
fp_table[0]=func_1;             /*initialize function*/
fp_table[1]=func_2;             /*pointer table*/
index=0;                        /*and function pointer index*/
...                             /*main program loop*/
index=fp_table[ index ]();      /*do appropriate function*/
...
}
```

Fig 3—You can speed up even pure C programs by replacing switch/case statements with a table of pointers to the control functions.

Transferring control by means of a table of function pointers is twice as fast as using C switch/case statements.

The remaining lines of code define the table of C function pointers (lines 143 through 174); some global variables (lines 176 through 183); and the control buffer (lines 184 through 188).

The time required for the folding-machine controller to process the single function from the entry of the processing loop at FDIR_050 (**Listing 2,** line 21) to the exit of the loop at FDIR_900 (line 127) is 216 μsec. Processing seven functions requires about 1250 μsec, depending on which functions are active. In contrast, seven executions of a single switch statement require 1750 μsec plus about 1 msec for housekeeping and actual function execution. It is apparent that the function-pointer method is at least twice as fast as the switch/case method.

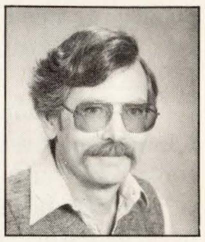
In a strictly C environment you might rewrite a switch statement and its associated code as indexed function-pointer addresses (**Fig 3**), since you would expect to see some speed improvement whenever you replace switch statements with indexed function-pointer addresses. Indeed, comparing the speed of the two methods on a personal computer shows that function-pointer control requires only 50 to 75% of the time required by switch/case control.

Programs written entirely or partly in C will probably never be as fast as an equivalent program written entirely in assembly language. However, to achieve an incremental speed increase without abandoning C, using indexed function pointers for program-flow control is a generic technique that you can adapt to many applications. EDN

Reference

1. Brown, Rick, 'C function pointers let you build generic control systems,' *EDN*, April 26, 1990, pg 215.

Author's biography



Rick Brown is an associate research engineer at the Desert Research Institute of the University of Nevada (Reno, NV), where he has worked for 18 years. Rick's prime responsibility is the development of instruments and instrumentation systems. He holds a BSEE from the University of Florida (Gainsville, FL) and is a member of the IEEE. He also runs a consulting service specializing in the design of embedded-processor systems. In his spare time he enjoys mountain climbing, skiing, and backcountry travel.

Article Interest Quotient (Circle One)
High 470 Medium 471 Low 472

Listing 1—Sample of C routines for folding-machine controller

```
/* wait for material arrival at sensor 1 */
fa_start( time )          /****** routine 0 ******/
int *time;
{
if( fa_stopped ) return(0);        /*no auto action when stop*/
if( !(FA_PDREG & B1PD) )
 {if( !fa_pd1clear ) return(0);    /*wait for sheet to leave*/
  fa_pd1clear=0;                   /*sheet in path*/
  sheet_count++;                   /*count arrivals*/
  *time=0;
  return(2);}                      /*to wait for pick up*/
fa_pd1clear=1;                     /*path is clear*/
return(0);
}
/* delay to pick up at station 1 */
fa_b1pick( time )        /****** routine 2 ******/
int *time;
{
```

Listing 1—Sample of C routines for folding-machine controller *(continued)*

```
if( !(FA_SENSE & B1BYPASS) )          /*low is true*/
 {*time = -b1b2_travel;
  return(8);}                         /*wait for arrival at b2*/
if( *time < b1apt ) return(2);        /*not yet*/
*time=0;
FA_CTLRG |= B1ACT;                    /*pick up*/
return(4);                            /*to wait for tail*/
}
/* wait for material tail at sensor 1 */
fa_b1tail( time )       /****** routine 4 ******/
int *time;
{
if( !(FA_PDREG & B1PD) ) return(4);       /*still in path*/
*(time+1) = *time;                    /*pick to tail*/
*time=0;
return(6);                            /*wait to drop*/
}
/* delay to drop at station 1 */
fa_b1drop( time )       /****** routine 6 ******/
int *time;
{
if( *time < b1btime ) return(6);      /*not time yet*/
FA_CTLRG &= ~B1ACT;
return(8);                            /*wait for arrival at b2*/
}
/* wait for material at sensor 2 */
fa_b2wait( time )       /****** routine 8 ******/
int *time;
{
if( FA_PDREG & B2PD ) return(8);      /*not there yet*/
*time=0
return(10);                           /*wait to pick*/
}
```

Listing 2—Assembly-language function driver for folding-machine controller

```
;
;           registers transferred to/from hardware
;
        PUBLIC  FA_CTLRG,FA_MOTRG,FA_SENSE,FA_PDREG
;
        NAME    FOLD_DIR
        CSEG
;
FOLD_DIR:       LD      BC,1001H        ;READ PHOTO DETECTORS
                IN      A,(C)
                LD      (FA_PDREG),A
                LD      C,5             ;READ SWITCHES
                IN      A,(C)
                LD      (FA_SENSE),A
                CALL    IRON_SPD        ;MONITOR IRONER SPEED
                LD      A,(FN_SKBOT)    ;CONTROL BUFFR BOTTOM
                AND     7               ;CIRCULAR 8 POSNS LONG
                LD      C,A             ;BUFFR COUNTER IN BC
                LD      B,0
                                       ;ALWAYS RUN BOTTOM BUFFR POSN
FDIR_050:       PUSH    BC              ;CURRENT BUFFR COUNTER
```

Listing continued

Listing 2—Assembly-language function driver for folding-machine controller ***(continued)***

```
            LD      HL,FN_BUFFR     ;CIRCULAR BUFFR BASE ADDR
            LD      A,C             ;+ 5*CONTROL BUFFR CTR
            ADD     A,C             ;TO PROCESS BUFFR FROM
            ADD     A,C             ;BOTTOM TO TOP
            ADD     A,C
            ADD     A,C
            LD      C,A
            ADD     HL,BC           ;POINTS TO CURRENT ENTRY
            PUSH    HL              ;IN CIRCULAR BUFFR
            LD      E,(HL)          ;(TIME)
            INC     HL
            LD      D,(HL)
            INC     DE              ;INCREMENT TIME
            POP     HL
            PUSH    HL
            LD      (HL),E          ;REPLACE TIME
            INC     HL
            LD      (HL),D
            POP     HL              ;ADDRESS OF TIME IN BUFFR
            LD      B,H             ;SAVE TO PASS TO ROUTINE
            LD      C,L
            INC     HL
            INC     HL
            INC     HL
            INC     HL              ;POINT TO INDEX CONTROL
            PUSH    HL              ;SAVE TO REPLACE INDEX
            LD      DE,FDIR_075     ;RETURN ADDRESS
            PUSH    DE              ;SAVE FOR 'RET' INSTR
            LD      D,0
            LD      A,(HL)          ;GET ROUTINE INDEX
            AND     3EH             ;MASK TO 32 X 2 ARRAY
            LD      E,A
            EX      DE,HL           ;IN HL
            LD      DE,FN_VECT      ;FUNCTION PTR TABLE ADDR
            ADD     HL,DE           ;ADDRESS OF FUNCTION PTR
            LD      E,(HL)          ;C FUNCTION ADDRESS
            INC     HL              ;TO CALL BY 'RET'
            LD      D,(HL)
            PUSH    DE              ;ON BUFFR TO EXECUTE
            LD      H,B             ;PASS CONTROL BUFFR ADDR
            LD      L,C             ;TO FUNCTION
            RET                     ;GO TO FUNCTION
;
FDIR_075:   LD      A,L             ;INDEX RETURNED
            POP     HL              ;ADDRESS OF INDEX
            LD      (HL),A          ;REPLACE INDEX
            POP     BC              ;CURRENT BUFFR COUNTER
            CP      0               ;TEST INDEX FOR INACTIVE
            JR      NZ,FDIR_200     ;IS ACTIVE
                                    ;IS INACTIVE
            LD      A,(FN_SKTOP)    ;IF CURRENT=TOP
            CP      C               ;WAITING FOR NEW MATERIAL
            JP      Z,FDIR_900      ;AND TEST IS COMPLETE
;
            LD      A,(FN_SKBOT)    ;IF CURRENT=BOTTOM THEN
            CP      C               ;MATERIAL COMPLETE
            JR      NZ,FDIR_080     ;IF NOT THEN SEQ ABORTED
            INC     A               ;SO MOVE UP BOTTOM
```

Listing 2—Assembly-language function driver for folding-machine controller *(continued)*

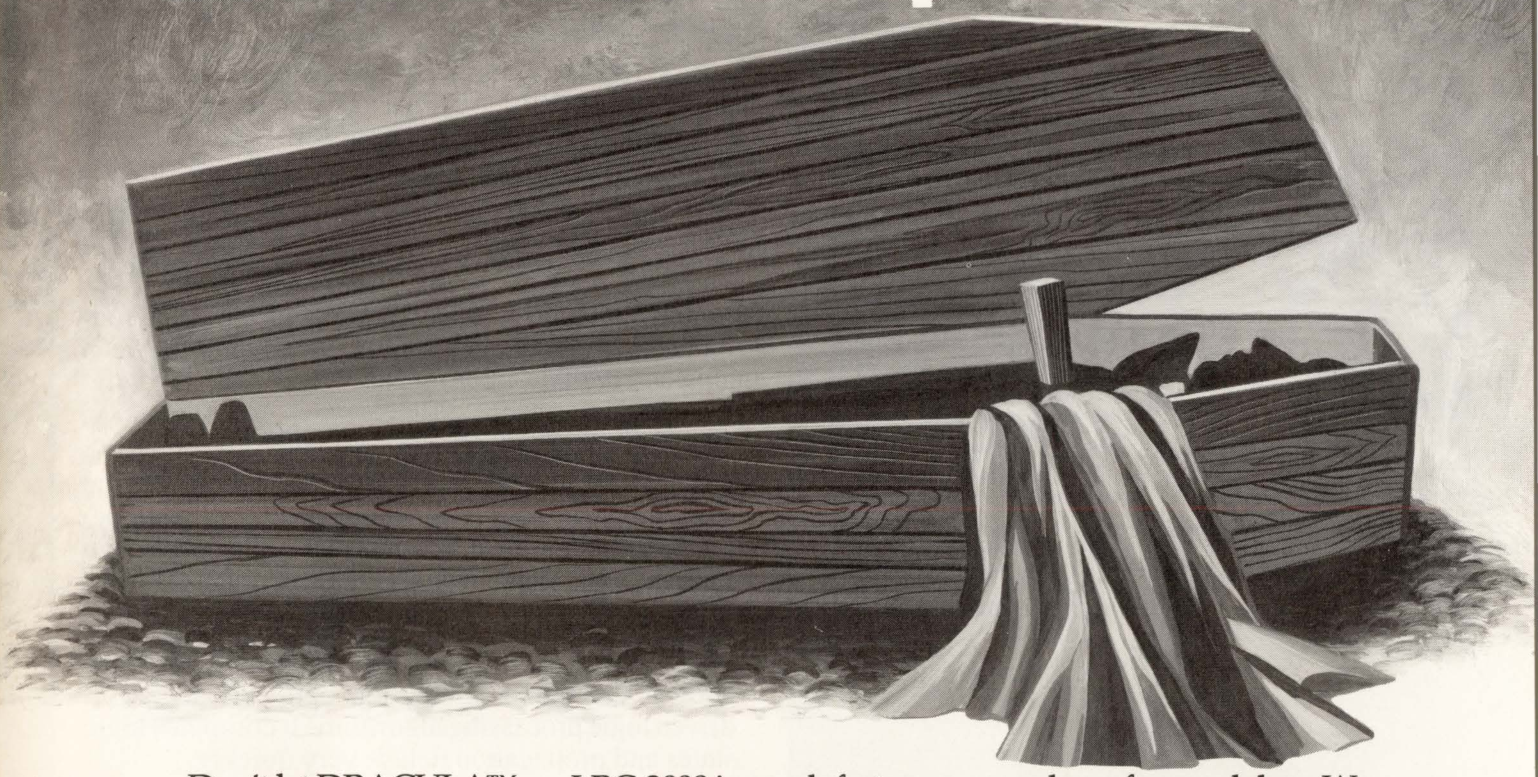
```
                AND     7                ;CIRCULAR 8 POSNS LONG
                LD      (FN_SKBOT),A     ;NEW BUFFR BOTTOM
                JR      FDIR_500         ;CONTINUE
FDIR_080:       LD      A,62             ;CALL FOR POSITION CLEAR
                LD      (HL),A           ;UNTIL AT BUFFR BOTTOM
                JR      FDIR_500         ;AND CONTINUE
                                         ;INDEX .NE. 0
FDIR_200:       LD      A,(FN_SKTOP)     ;TEST FOR NEW
                CP      C                ;BY CURRENT == TOP
                JR      NZ,FDIR_500      ;NOT NEW..CONTINUE
                INC     A                ;IS NEW..INCREASE BFR TOP
                AND     7                ;CIRCULAR BUFFR
                LD      (FN_SKTOP),A     ;NEW BUFFR TOP
                                         ;
                LD      C,A
                LD      A,(FN_SKBOT)     ;TEST FOR BOTTOM
                CP      C                ;OVER WRITTEN
                JR      NZ,FDIR_250
                CALL    EXCEPTION        ;BUFFER OVERFLOW
;
FDIR_250:       LD      A,(FN_SKTOP)     ;NEW CONTROL SEQUENCE
                AND     7                ;BY PLACING ENTRY MONITOR
                LD      C,A              ;ROUTINE AT BUFFR TOP
                LD      B,0
                ADD     A,C              ;BRIDGE 1 PD
                ADD     A,C              ;INDEX MUST BE 0
                ADD     A,C              ;POINT TO CONTROL TABLE
                ADD     A,C              ;LOCATION 0
                LD      C,A              ;THEN CALCULATE ADDRESS
                LD      HL,FN_BUFFR      ;OF TIME WORD
                ADD     HL,BC
                LD      (HL),0           ;TIME = 0
                INC     HL
                LD      (HL),0
                INC     HL
                INC     HL
                INC     HL               ;ADDRESS OF INDEX
                LD      (HL),0           ;INDEX = 0
                JP      FDIR_900         ;END OF INTERRUPT
FDIR_500:                       ;CONTINUE ON TO END OF BUFFR
                INC     C                ;UPDATE CURRENT
                LD      A,7              ;BUFFR COUNTER
                AND     C                ;MASK TO CIRCULAR
                LD      C,A
                LD      B,0
                JP      FDIR_050         ;DO NEXT BUFFR LOCATION
;
FDIR_900:       LD      BC,1004H         ;OUT TO MOTORS
                LD      A,(FA_MOTRG)
                OUT     (C),A
                LD      C,6              ;OUT TO CONTROL
                LD      A,(FA_CTLRG)
                OUT     (C),A
                RET                      ;TO INTERRUPT PROCESSOR
;
;  ROUTINE TO CLEAR BUFFR POSITION
;
```

Listing continued

Listing 2—Assembly-language function driver for folding-machine controller *(continued)*

```
FA_CLEAR:       LD      HL,0
                RET
;
;
; FUNCTION POINTER TABLE.  POINTED TO BY INDEX
;
FN_VECT:        DW      FA_START        ; 0 TEST FOR START
                DW      FA_B1PICK       ; 2 WAIT FOR B1 PICK UP
                DW      FA_B1TAIL       ; 4 WAIT FOR B1 TAIL
                DW      FA_B1DROP       ; 6 WAIT TO DROP
                DW      FA_B2WAIT       ;WAIT FOR ARRIVAL AT B2
                DW      FA_B2PICK       ;B2 PICK UP
                DW      FA_B2TAIL       ;12
                DW      FA_B2DROP       ;B2 DROP
                DW      FA_C1WAIT       ;WAIT FOR ARRIVAL AT C1
                DW      FA_C1BREAK      ;BREAK MOTOR STOP
                DW      FA_C1AON        ;20 START AIR BLAST
                DW      FA_C1AOF        ;22 STOP AIR BLAST
                DW      FA_C2WAIT       ;WAIT FOR ARRIVAL AT C2
                DW      FA_C2PICK       ;C2 PICK UP
                DW      FA_C2TAIL       ;28
                DW      FA_C2DROP       ;C2 DROP
                DW      FA_C3WAIT       ;WAIT FOR ARRIVAL AT C3
                DW      FA_C3PICK       ;C3 PICK UP
                DW      FA_C3TAIL       ;36
                DW      FA_C3DROP       ;C3 DROP
                DW      FA_BYCROSS      ;BYPASS CROSS FOLDS
                DW      FA_B2BYTAIL     ;TAIL AT B2
                DW      FA_B2CLOSE      ;CLOSE BYPASS CHUTE
                DW      FA_BUZZER       ;SOUND THE HORN
                DW      FA_TANGLE       ;STOP FOR POSSIBLE TANGLE
                DW      FA_CLEAR        ;50 DUMMY LOCATION
                DW      FA_CLEAR        ;DUMMY LOCATION
                DW      FA_CLEAR        ;DUMMY LOCATION
                DW      FA_CLEAR        ;DUMMY LOCATION
                DW      FA_CLEAR        ;DUMMY LOCATION
                DW      FA_CLEAR        ;DUMMY LOCATION
                DW      FA_CLEAR        ;62 DUMMY LOCATION
;
        DSEG
;
FA_CTLRG:       DB      1               ;CONTROL SIGNALS
FA_MOTRG:       DB      1               ;MOTOR CONTROL
FA_SENSE:       DW      1               ;
FA_PDREG:       DW      1               ;PHOTO DETECTORS
FN_SKBOT:       DB      1               ;BOTTOM OF BUFFR POINTER
FN_SKTOP:       DB      1               ;TOP OF BUFFR POINTER
;                            PARAMETER BUFFER FOLLOWS
FN_BUFFR:       DW      1               ;TIME WORD
                DW      1               ;AUX TIME WORD
                DB      1               ;ENT BYTE
                DB      35              ;REST OF BUFFER
                END

XXXXXX
22 hours
XXXXXX
```

ISS. We've driven a stake through the DRC vampire



Don't let DRACULA™ hypnotize you into a dark corner. Before you choose a design rules checker, let ISS shed some light on the subject. We have DRC software in our design toolkit that's worth evaluating...

It's called LRC-2000™, and it's a true hierarchical design rules checker. LRC-2000 is much faster than the vampire. And we pack it with more functionality–like reduced false errors, parallel processing, and a one-for-one conversion of DRACULA run sets.

With all these improvements over the DRC vampire, you might think our DRC costs more. The truth is you can buy ISS 's DRC package for much less. We won't bleed you dry.

So why not exorcise *your* vampire and try ISS's LRC-2000 free for 30 days. Install it on the workstation of your choice, or visit one of our sales offices for a demonstration.

You'll feel alive again! Call us toll free at—

1-800-4-CAD-LTL

ISS *Integrated Silicon Systems, Inc.*

ISS. The IC CAD company that listens.

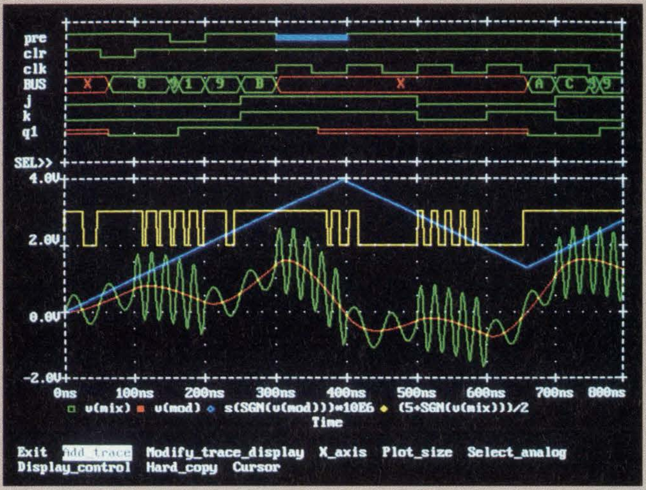
CIRCLE NO. 15

P.O. Box 13665, Research Triangle Park, NC 27709 Phone: 919/361-5814 Fax: 361-2019
Silicon Valley: 408/562-6154 S. California: 714/891-0203 Texas: 512/452-5814

PSpice

MicroSim Corporation

The Standard for Circuit Simulation



Analog and digital waveforms with common time axis

MicroSim is the leader in Mixed Analog/Digital Simulation Technology

CIRCLE NO. 11

Two years ago, MicroSim introduced the technology for simulating mixed analog/digital circuits. Now, this capability is available in the Digital Simulation extension for PSpice. It does true mixed-mode simulation of circuits – including feedback loops between analog and digital sections.

PSpice's Digital Simulation option performs mixed analog and digital simulations. There are no performance compromises–digital components are processed at logic simulation speeds and analog waveforms are calculated with PSpice's usual precision. Analog and digital waveforms may be displayed together, with a common time axis (see photograph for an example).

Digital Simulation removes one of the greatest constraints on circuit simulation: the dichotomy between analog and digital.

PSpice with the Digital Simulation option:

- Is Easy to Use – In the circuit description, digital devices follow the same syntax as other PSpice devices. Simulating a mixed circuit is no different from running a transient analysis on an analog circuit.
- Has Efficient Algorithms – To get reasonable speed–and to allow reasonably large sections of digital circuitry–the core of theDigital Simulation option is an event-driven logic processing algorithm. It computes logic states and propagation delays very quickly.
- Includes an Extensive Library – The Digital Simulation option includes libraries for most TTL components. These include gates, flip-flops, latches, registers, and counters. Each component, in turn, includes models for many logic families including TTL, LS, ALS, H, F, L, S, AS, HC, HCT, and 4000 series CMOS. The component models describe not only the functionality of the device but also all its propagation delays.

Each copy of PSpice comes with our extensive product support. Our technical staff has over 100 years of experience of CAD/CAE, and our software is supported by the engineers who wrote it.

For further information about the Digital Simulation option or any other PSpice product, please call us **toll free** at (800) 826-8603 or, in California, (714) 770-3022. Find out for yourself why PSpice has sold more programs than all other SPICE-type programs combined and has become the de facto standard for circuit simulation.

20 Fairbanks • Irvine, CA 92718 USA • Telex 265154 SPICE UR

Comprehensive features ease ports from Unix to OS/2

Porting an application from one operating system to another is rarely a simple procedure. Differences in system features and operating limits complicate the port. However, in the case of porting Unix applications to OS/2, your task is simpler because OS/2 has features similar or superior to Unix.

Joseph Gnocato, *MPR Teltech Ltd*

The problems peculiar to porting a Unix application to OS/2 stem from the substantial differences between the architectures of the two operating systems. The differences have a profound effect on how you write your code, how you structure your programs, and how you use your compiler.

A standard Unix application has difficulty processing the message-based "events" of OS/2. Under OS/2's graphical user interface, Presentation Manager (PM), all user-input events send messages to the active application via a queue. The user-input events first arrive at a System Message Queue, which subsequently sends them to the active application. The active application must always be ready to respond to any of its messages at any time.

For example, if an OS/2 PM application expects some keyboard input from the user, among other inputs, it cannot merely call the C function *getchar*. The *getchar* function call would "block" the application and wait for the user's input. If the user does not type a character, but instead tries to resize the window, the application cannot handle the window-resize message because the program is blocked waiting for user input.

Although the message-based architecture of PM seems to create a problem for software engineers who want to use functions like *getchar*, OS/2 provides a powerful solution. It allows a "process," or "task," to have more than one "thread" of execution. Threads can execute different parts of a single program concurrently. This concurrency means that one thread can manage window-manipulation tasks while another executes blocking functions like *getchar*.

In contrast, Unix-based applications generally do not operate in a message-based environment. Porting graphical Unix applications to PM is not simply a matter of blindly converting the graphics and file-system routines to equivalent PM routines. You must carefully consider how the Unix application's existing user-event handler will interact with OS/2 PM. Usually, your user-event handling will require some modification.

Handler acquires semaphores

Unix applications may require several fundamental changes to handle events under PM. You must employ semaphores to control different threads and to serialize specific actions. For example, an application that must periodically recalculate a large spreadsheet must still respond to the user during recalculation.

As a simple example, consider a program that begins a time-consuming operation whenever the user presses

The existence of virtual memory on the target computer is usually a prerequisite for successfully porting any large application.

Listing 2—MTDEMO definitions file

```
NAME            MTDEMO  WINDOWAPI

DESCRIPTION     'MTDEMO Program, (c) Joe Gnocato, 1990'
PROTMODE
HEAPSIZE        1024
STACKSIZE       8192
EXPORTS         ClientWndProc
```

the second mouse button. Now, assume that a user first presses the second mouse button and then decides to resize the program's window. In a single-threaded PM program, the window resizing would not occur until after the time-consuming operation was completed. Alternatively, a multithreaded program performs the time-consuming operation and the window resizing concurrently.

Listing 1 (see pg 59) contains a complete multithreaded, semaphore-controlled program for OS/2 PM called MTDEMO. **Listings 2** and 3 show the module-definition file and the make file, respectively, for the MTDEMO program. MTDEMO demonstrates how threads and semaphores combine to provide effective system response to the user.

In the PM code, the *main* procedure initializes the program's environment and creates a Frame and Client window on the PM Desktop (Presentation Manager's metaphor for its screen displays). Before entering the event-processing loop, *main* creates a second thread of execution with the *_beginthread* function.

The *main* procedure then starts processing events by repeatedly calling the *WinGetMsg* and *WinDispatchMsg* functions. When the user closes the program's window, *WinGetMsg* returns FALSE, and the *main* program terminates.

Simultaneously, the *_beginthread* function begins executing the second thread with the *SecondThreadFct* procedure. This procedure is an infinite loop that sets the *hsemPauseThread2* semaphore and waits for it to be cleared. When *hsemPauseThread2* clears, a beep sounds and the time-consuming operation begins (in this case a *sleep* operation). When it ends, a beep sounds again. The second thread sets the semaphore, and then waits for it to clear again.

The function *ClientWndProc* clears the *hsemPauseThread2* semaphore upon receiving the *WM_BUTTON2DOWN* message. This message occurs whenever the user presses the second mouse button.

Assume that a user invokes MTDEMO, presses the second mouse button, moves to the window border, and then resizes the window. **Fig 1** shows a timing diagram for a single-threaded version of MTDEMO. Notice that all processing occurs serially. Compare these results with **Fig 2**, which shows the timing diagram for the two-threaded version of MTDEMO.

A single-threaded implementation of MTDEMO would not create a second thread; it would invoke the time-consuming operation directly from *ClientWndProc* upon receiving the *WM_BUTTON2DOWN* message. Thus you can use multiple threads to handle several events in a seemingly simultaneous manner. Imagine an application that waits for user input, resizes its window, redraws the graphics area, and loads a data file—all simultaneously. OS/2's threads make this concurrency possible. Applications using threads will become even more powerful on architectures that have multiple CPUs, where each CPU will have one or more threads.

OS/2 supports a "DOS-style" file system that re-

Listing 3—MTDEMO make file

```
mtdemo.obj : mtdemo.c
     cl -c -AH -G2sw mtdemo.c

mtdemo.exe : mtdemo.obj mtdemo.def
     link mtdemo.obj, /align:16, NUL, /NOD llibcmt os2, mtdemo.def
```

stricts file names to an 8-character name followed by a period separator ("dot") and a 3-character extension. Unix users often find the DOS-style naming convention restrictive. When porting from Unix to OS/2, you must ensure that each application conforms to the DOS-style file-naming conventions.

With the introduction of OS/2 version 1.2, OS/2 began supporting multiple, installable file systems. The High Performance File System (HPFS) is the new OS/2 file system that removes the DOS-style file naming restrictions. HPFS file names may be very long strings composed of any combination of legal characters. This new file system removes many of the naming incompatibilities present between the Unix and DOS-style file systems.

Accommodating the differences between Unix and OS/2 often comes down to compiler considerations. For example, software engineers developing Unix applications never worry about which memory model to use during application development—Unix implicitly assumes a "flat" memory architecture. In contrast, because of the peculiarities of the 80X86 family's architecture, OS/2 PM developers face the task of first choosing an appropriate memory model for their programs.

Standardizing on the *HUGE* memory model eliminates many of the precautions needed when mixing and matching object files and library programs compiled with different memory models. Using the *HUGE* memory model also means that the code does not require the *FAR* and *NEAR* keywords (available in the Microsoft C compiler). In addition, the *HUGE* memory model provides a flat memory model that most closely resembles that of Unix.

The existence of virtual memory on the target computer is usually a prerequisite for successfully porting any large application. OS/2's virtual memory handles all requests for dynamic memory and performs code or data swapping automatically. Program overlays, common in large DOS applications, are unnecessary under OS/2.

OS/2's virtual memory easily handles requests for large pieces of dynamically allocated memory. However, you will that the *halloc* function, in Microsoft's C runtime library V5.10, will not grant requests of more than 4M bytes per call. The *DosAllocHuge* function circumvents this restriction. **Listing 4** provides a source-code fragment that allocates 5M bytes of memory.

The *swapper.dat* file (usually \OS2\SYSTEM\SWAPPER.DAT) is OS/2's virtual-memory file on the hard disk. As requests for memory exceed the machine's RAM, the *swapper.dat* file grows dynamically as the RAM and hard disk swap data.

Unfortunately, the *swapper.dat* file does not dynamically shrink as an application frees memory. Users may see a sharp decrease in available hard-disk space when an application allocates large pieces of memory. The *swapper.dat* file only returns to its default size when users reboot their systems.

As with all ports, pay particular attention to indiscriminate mixing, or casting, of variables. Also pay attention to compiler-dependent "clever" programming. For example, in some C compilers for particular

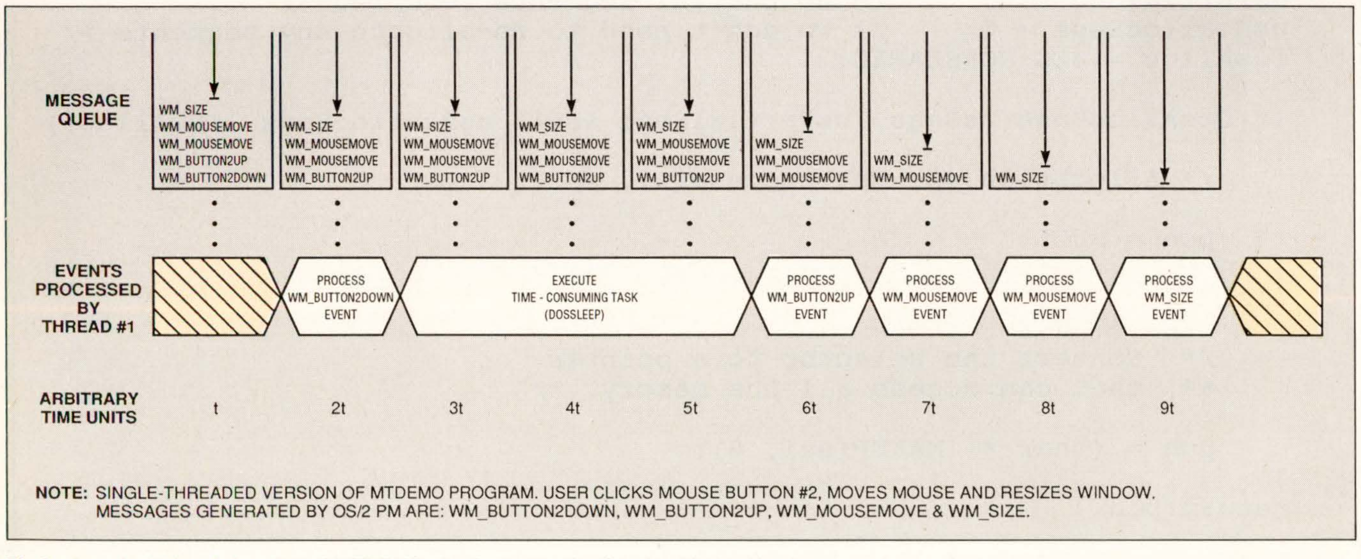

MESSAGE QUEUE
WM_SIZE
WM_MOUSEMOVE
WM_MOUSEMOVE
WM_BUTTON2UP
WM_BUTTON2DOWN
WM_SIZE
WM_MOUSEMOVE
WM_MOUSEMOVE
WM_BUTTON2UP
WM_SIZE
WM_MOUSEMOVE
WM_MOUSEMOVE
WM_BUTTON2UP
WM_SIZE
WM_MOUSEMOVE
WM_MOUSEMOVE
WM_BUTTON2UP
WM_SIZE
WM_MOUSEMOVE
WM_MOUSEMOVE
WM_BUTTON2UP
WM_SIZE
WM_MOUSEMOVE
WM_MOUSEMOVE
WM_SIZE
WM_MOUSEMOVE
WM_SIZE
EVENTS PROCESSED BY THREAD #1
PROCESS WM_BUTTON2DOWN EVENT
EXECUTE TIME - CONSUMING TASK (DOSSLEEP)
PROCESS WM_BUTTON2UP EVENT
PROCESS WM_MOUSEMOVE EVENT
PROCESS WM_MOUSEMOVE EVENT
PROCESS WM_SIZE EVENT
ARBITRARY TIME UNITS
t
2t
3t
4t
5t
6t
7t
8t
9t
NOTE: SINGLE-THREADED VERSION OF MTDEMO PROGRAM. USER CLICKS MOUSE BUTTON #2, MOVES MOUSE AND RESIZES WINDOW. MESSAGES GENERATED BY OS/2 PM ARE: WM_BUTTON2DOWN, WM_BUTTON2UP, WM_MOUSEMOVE & WM_SIZE.

Fig 1—In a single-threaded version of MTDEMO, the time-consuming function blocks other input-event processing.

Accommodating the differences between Unix and OS/2 often comes down to compiler considerations.

processors, an *int* may be the same size as a pointer. Code that implicitly makes this assumption could be difficult to port to other processors with different underlying architectures.

Useless C runtime functions

Under Presentation Manager, several standard C runtime I/O functions (such as *printf* and *getchar*) are no longer useful because the application must receive user input from the message queue. For each character the user types, the application will receive a message providing the key's code and the status of other keys. The application must process these messages and concatenate the key events to produce a string. You can use PM's Dialog Boxes to perform this task automatically.

The C function *printf* prints to *stdout*, but in a graphical PM program, *stdout* is not linked to the graphics window. Therefore, *printf* will not produce any visible result on a graphical program's PM screen. PM programs usually accomplish text output through PM's *Gpi* or *Vio* routines.

Therefore, instead of using the standard *printf* function throughout your application's source code, you should instead use a system-independent function, perhaps *MyPrintFunction*. The actual code for *MyPrintFunction* will change from system to system, but the function isolates the application from these changes.

Listing 4—OS/2 megabyte allocator

```
char * GiveMeFiveMB()
{
   /*  This function tries to allocate 5 MB of memory. It returns a
   **   pointer to the memory if successful, otherwise NULL.
   **
   **  You must compile this function with the LARGE or HUGE
   **  memory model options. */

   USHORT  usSegs, usPartialSeg;
   USHORT  usReallocSegs, fusAlloc;
   SEL     sel;
   CHAR    *pch;

   usSegs = 80;            /* 80 segments of 64KB = 5 MB */
   usPartialSeg = 0;       /* no partial segments required */
   usReallocSegs = 0;      /* we don't need to re-allocte any segments */
   fusAlloc = SEG_NONSHARED;

   if(DosAllocHuge(usSegs, usPartialSeg, &sel, usReallocSegs, fusAlloc))
   {
       /* Allocation was not successful. */

       pch = NULL;
   }
   else
   {
       /*  Convert the selector to a pointer
       **  that can access all the memory. */

       pch = (char *) MAKEP(sel, 0);
   }
   return(pch);
}
```

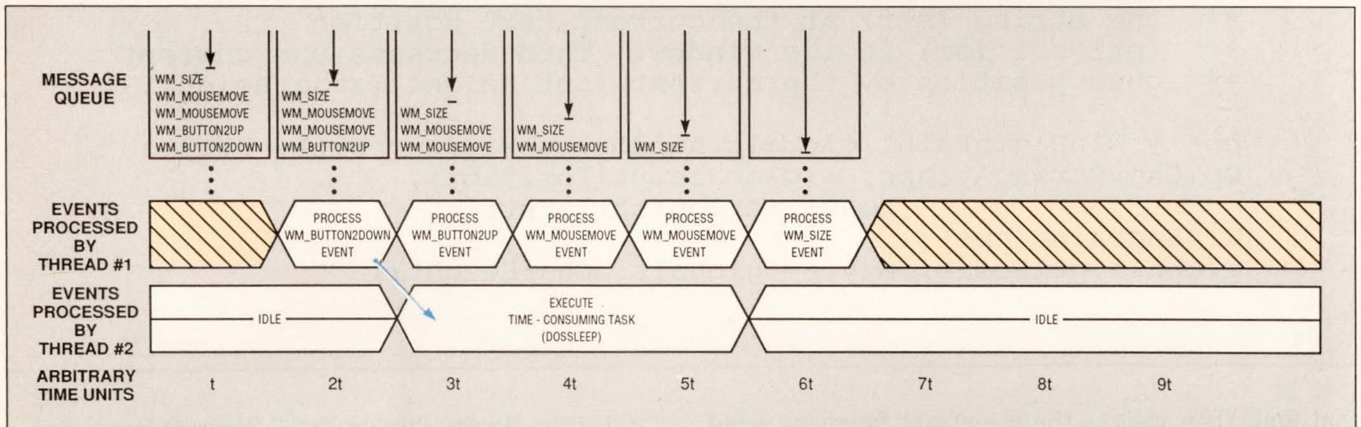
Listings 5a and **5b** show examples of Unix and OS/2 versions, respectively, of the actual code for *MyPrintFunction.*

Because of *MyPrintFunction*, the application is unaware of the actual mechanism for writing to the screen. Isolating the application from system dependencies minimizes the work required to port the application. In addition, environments that permit multiple fonts or colors can provide these features without making an impact on the application.

For multithreaded programs, where more than one thread makes use of the C runtime functions, you must use a special version of the Microsoft C runtime library. For example, suppose that one thread is in the middle of a *printf* call and the OS/2 scheduler suspends its execution. Now a second thread begins executing and it also makes a call to *printf.* With the regular (nonreentrant) function, the second thread's call to *printf* could corrupt the function's static data.

The Microsoft C compiler includes a large-model library that you should link to multithreaded programs where more than one thread executes C runtime functions. Applications do not require the multithreaded C runtime library when only one thread accesses the runtime library, or if the library accesses are done serially.

Note that some of the C runtime functions in the nonmultithreaded libraries are re-entrant. Programs



NOTE: TWO-THREADED VERSION OF MTDEMO PROGRAM.
SAME MESSAGES AS FIGURE #1 ARE QUEUED BY USER ACTIONS. HOWEVER, PROGRAM HANDLES THEM IN A DIFFERENT MANNER.
SECOND THREAD WORKS ON THE 'TIME-CONSUMING' TASK WHILE FIRST THREAD CAN IMMEDIATELY RESPOND TO USER'S 'RESIZE' REQUEST.

Fig 2—The two-threaded version of MTDEMO can simultaneously execute both the time-consuming task and service the user's mouse events.

Points to ponder when porting programs

As OS/2 continues to mature, more software engineers will port Unix applications to Presentation Manager and OS/2. These software engineers should consider the following points:

- You will need to tailor your Unix application's user-event handler for operation under OS/2 and Presentation Manager (PM).
- You will need multiple threads in the OS/2 PM application especially if the application uses blocking functions.
- A system-independent alternative for *printf* is a necessity.
- Applications with multiple threads, calling the C runtime library, require special multithreaded versions of the C runtime library programs.
- The possible incompatibilities in the file-naming conventions between the Unix file system and OS/2's DOS-style file system are often troublesome. However, OS/2's new High Performance File System (HPFS) eliminates most of these concerns.
- OS/2's virtual memory provides elegant memory management and is transparent to the user.
- Prototyping menus, dialog boxes, and icons under OS/2 PM is very quick and easy.

Unix users often find the DOS-style naming convention restrictive.

Listing 5—Unix and OS/2 versions of a system-independent printing function

```
void MyPrintFunction(str)
char    *str;
{
    printf("%s\n", str);
}
```

(a)

```
void MyPrintFunction(str)
char    *str;
{
    HPS        hps;

    /*  Obtain a Presentation Space handle (hps) and print
    **  the string (str) at the current text position
    **  (ptlPosition) in the window.  Then decrease the current
    **  text position by the current font height (sCharHeightY). */

    hps = WinBeginPaint(globals.hwndOutputWindow, NULL, NULL);
    GpiCharStringAt(hps, &(globals.ptlPosition),
                    (LONG) strlen(str), str);
    WinEndPaint(hps);
    globals.ptlPosition.y -= globals.sCharHeightY;
}
```

(b)

that limit their calls to the re-entrant functions could, therefore, use the regular functions. However, in practice, worrying about which functions can or cannot be used for multithreaded programs is too troublesome. Standardizing on the multithreaded libraries eliminates the need to check all code for calls to nonre-entrant C functions.

The OS/2 utilities for developing menus, dialog boxes, and icons are very useful and allow rapid prototyping of several aspects of your user interface.

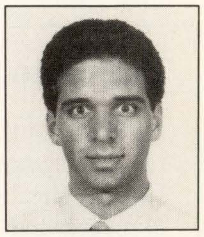
Using the PM's Resource-Description Language, you can describe user menus and dialog boxes with text. The resource compiler compiles your descriptions into binary files that "bind" with your executable program. Alternatively, you can interactively construct dialog boxes with the *DlgBox* utility provided in the Software Developers' Kit. EDN

References

1. Petzold, Charles, "Programming, the OS/2 Presentation Manager." Microsoft Press, Redmond, WA, 1989. ISBN 1-55615-170-5.

2. Letwin, Gordon, "Inside OS/2." Microsoft Press, Redmond, WA, 1988. ISBN 1-55615-117-9.

Author's biography



Joseph Gnocato is a systems development engineer with MPR Teltech Ltd in Burnaby, British Columbia, Canada. He's been with the company for four years. He develops CAD/CAE applications and user interfaces under Unix and OS/2. Joseph earned a BASC in electrical engineering from the University of British Columbia. Joseph, a member of the IEEE, enjoys traveling and river rafting.

Article Interest Quotient (Circle One)
High 479 Medium 480 Low 481

Listing 1—Multithreaded demonstration program

```
/*  PROGRAM: mtdemo (mutliple thread demo)
**
**  PURPOSE: This OS/2 PM program demonstrates:
**             1) executing multiple threads
**             2) semaphores controlling threads */

#define INCL_WIN
#define INCL_DOS
#include <os2.h>
#include <mt\process.h>       /* multi-threaded 'process.h' header file */
#include <mt\stdlib.h>        /* multi-threaded 'stdlib.h' header file */

#define STACKSIZE             4096        /* stack size for 2nd thread */
#define SLEEP_DURATION        5000        /* 5000 milliseconds */
#define BEEP_DURATION         150         /* 150 milliseconds */
#define HI_FREQ_BEEP          4000        /* 4000 Hz */
#define LO_FREQ_BEEP          100         /* 100 Hz */

void                main(void);
void                SecondThreadFct(void);
MRESULT EXPENTRY    ClientWndProc(HWND, USHORT, MPARAM, MPARAM);

HSEM    hsemPauseThread2;         /* control semaphore for thread#2 */

void main(void)
{

   /*  This sample program waits for the user to press the second
   **  mouse button.  When the user presses the button, a low-frequency
   **  beep sounds and a 'time-consuming' task begins in a
   **  second thread.  When this task completes, a high-frequency
   **  beep sounds.  While the 'time-consuming' task executes,
   **  the first thread still responds to the user's mouse and
   **  keyboard inputs.
   **
   **  The routine 'main' initializes the PM environment and
   **  launches the second thread of execution with the
   **  '_beginthread' function.  The 'hsemPauseThread2' semaphore controls
   **  the second thread's execution.
   **
   **  The 'WinDispatchMsg' function dispatches Message-based events
   **  arriving at the window's queue. The 'ClientWndProc' function
   **  processes messages. */

   HAB           hab;
   HMQ           hmq;
   QMSG          qmsg;
   HWND          hwndFrame, hwndClient;
   ULONG         FrameFlags;
   static CHAR   szClientClass[] = "DemoProg";
   static CHAR   ThreadStack[STACKSIZE];
   static TID    Thread2ID;
```

Listing continued

Threads can execute different parts of a single program concurrently.

Listing 1—Multithreaded demonstration program *(continued)*

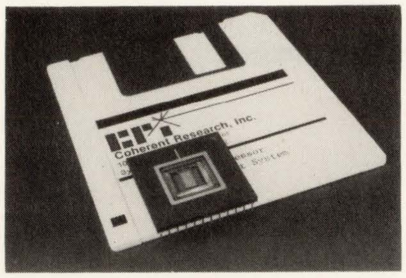
```
    /* Initialize PM and create a message queue for the events. */

    hab = WinInitialize(0);
    hmq = WinCreateMsgQueue(hab, 0);

    /*  Register the window class and create a standard
    **  frame window and client window. */

    WinRegisterClass(hab,                  /* anchor block handle */
                     szClientClass,        /* window class name */
                     ClientWndProc,        /* window procedure */
                     CS_SIZEREDRAW,        /* window-style flags */
                     0);                   /* amount of reserved data */

    FrameFlags = FCF_TITLEBAR | FCF_SYSMENU | FCF_SIZEBORDER |
                 FCF_MINMAX | FCF_SHELLPOSITION;

    hwndFrame = WinCreateStdWindow(HWND_DESKTOP,   /* parent window handle */
                                   WS_VISIBLE,     /* frame window style */
                                   &FrameFlags,    /* frame creation flags */
                                   szClientClass,  /* client class name */
                                   " Demo",        /* title-bar text */
                                   0L,             /* client window style */
                                   NULL,           /* resource module handle */
                                   NULL,           /* frame resource id */
                                   &hwndClient);   /* client window handle */

    /* Before beginning the event processing, start a second thread
    **  of execution. The second thread will execute the function
    **  called 'SecondThreadFct'. */

    Thread2ID = _beginthread(SecondThreadFct,      /* thread2 startup function */
                  (void *) ThreadStack,            /* thread2 stack space */
                  STACKSIZE,                       /* stack size for thread2 */
                  NULL);                           /* thread2 parameters */

    /* Loop through the user events until program halts. */

    while(WinGetMsg(hab, &qmsg, NULL, 0, 0))
    {
        WinDispatchMsg(hab, &qmsg);
    }

    /* Program has terminated, so clean up windows and queues. */

    WinDestroyWindow(hwndFrame);
    WinDestroyMsgQueue(hmq);
    WinTerminate(hab);
}

/*-----------------------------------------------------------------------*/

MRESULT EXPENTRY ClientWndProc(hwnd, msg, mp1, mp2)
HWND hwnd;
USHORT msg;
```

Listing 1—Multithreaded demonstration program *(continued)*

```
MPARAM mp1;
MPARAM mp2;
{
  /*  This function processes all Client window event messages.
  **  We are only interested in the message generated when
  **  the user presses the second mouse button. Let the system
  **  handle all other messages in the default manner. */

  switch(msg)
  {
    case WM_BUTTON2DOWN:

        /* Clear the semaphore to wake up the second thread.
        **  Then continue to process user events normally. */

        DosSemClear(&hsemPauseThread2);
        break;
  }
  return(WinDefWindowProc(hwnd, msg, mp1, mp2));
}

/*------------------------------------------------------------------------------*/

void SecondThreadFct(void)
{
  /*  Only the second thread executes this function.
  **  The 'hsemPauseThread2' semaphore controls execution.
  **
  **  This routine will SET the 'hsemPauseThread2' semaphore and
  **  then wait indefinitely until the semaphore clears.  When the user
  **  clears 'hsemPauseThread2' by pressing the second
  **  mouse button the routine wakes up and makes a low-freq beep.
  **
  **  The routine then simulates a 'time-consuming' task by sleeping
  **  for SLEEP_DURATION milliseconds.  The 'time-consuming' task would
  **  typically be the recalculation of a large spreadsheet or waiting
  **  for user input.
  **
  **  After the 'time-consuming' task has completed, the routine makes
  **  a high-freq beep and reenters a dormant state by setting the
  **  'hsemPauseThread2' semaphore and then waiting for it to clear. */

  while(TRUE)
  {
    /*  Set the semaphore and wait for it to clear.  Pressing the
    **  second mouse button will clear the semaphore, (see the
    **  'ClientWndProc' procedure). */

    DosSemSet(&hsemPauseThread2);
    DosSemWait(&hsemPauseThread2, SEM_INDEFINITE_WAIT);

    DosBeep(LO_FREQ_BEEP, BEEP_DURATION);
    DosSleep((ULONG) SLEEP_DURATION);   /* perform time-consuming task */
    DosBeep(HI_FREQ_BEEP, BEEP_DURATION);
  }
}
```

NEW PRODUCTS

SOFTWARE DEVELOPMENT TOOLS



Software Tool For Parallel Processors

- *Lets you write/debug programs for a parallel system*
- *Provides for use of associative memory*

The Coherent Processor (CP) Simulator package is a development system that lets you write, debug, and simulate the execution of application programs that will eventually run on the vendor's CP. The CP is a hardware parallel-processing system that attaches to a PS/2. Each processing element of the CP has 36 bits of content-addressable memory, as well as general-purpose registers. An interconnection network links the processing elements together in an SIMD (single-instruction, multiple-data) configuration. The simulation software runs on IBM PS/2s and compatibles and on Sun workstations, and it accurately models the hardware. The package consists of an assembler and linker, the simulator itself, and the CP Debugger, which gives you source-level debugging facilities. You write your program in CP assembly language and C. The assembler generates C source code representing the corresponding CP microcode. You then compile the entire program with the Microsoft C compiler and link it to the simulator, which the vendor supplies in the form of a library of object-code routines. When you load and run the resulting executable program, the simulator routines execute your application exactly as if it were running on the hardware. Each memory model has a separate library (small, medium, and large); you select the appropriate library for your application. The simulator package costs $895.

Coherent Research Inc, 1 Adler Dr, East Syracuse, NY 13057. Phone (315) 433-1010.

Circle No. 351

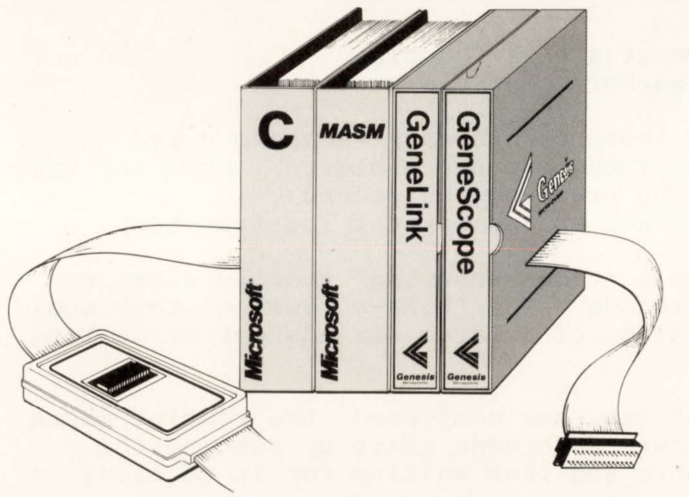
ANSI-C Compiler For MS-DOS

- *Conforms fully to the ANSI C standard*
- *Compatible with pre-ANSI-C implementations*

New C is a fully conforming ANSI-C compiler that runs on 80386-based computers under MS-DOS. New C is compatible with older implementations such as those based on Kernighan & Ritchie and the Berkeley 4.2 Portable C Compiler. You can therefore use New C to compile existing C code without any coding changes. The compiler also supports some non-ANSI extensions. It runs in protected 32-bit mode and uses the MS-DOS operating system for all I/O and system calls. Further, it lets you address all of the computer's physical memory at one time—it doesn't require you to conform to the 80286 memory-segmentation scheme. The compiler comes with a preprocessor, a standard runtime library with header files, and function prototypes. It provides optional in-line code generation of math and string-handling functions. $495.

Language Processors Inc, 959 Concord St, Framingham, MA 01701. Phone (508) 626-0006.

Circle No. 352



Quality ROM Tools

Genesis Microsystems has been a producer of quality 8086-family ROM development tools since 1982.

Our GeneLink linker/locator links OBJ files from Microsoft-C, MASM, and other languages to fully-located ABS and HEX files. CodeView debugging information is completely converted to standard Intel format for emulators and ROM debuggers. GeneLink is by far the fastest and most integrated ROM linker available, performing all operations with a single, integrated tool.

Our GeneScope/Target PROM source-level target debugger gives you the same advanced, windowed debugging capabilities as we supply to premium emulator manufacturers for their own high-end products, but at the lower costs of a PROM-based monitor.

For more information call Genesis Microsystems Corp. at (707) 542-5000. Or write 146-D Wikiup Dr., Santa Rosa, CA 95403.

CIRCLE NO. 23

EDN PRODUCT MART

This advertising is for new and current products.

Please circle Reader Service number for additional information from manufacturers.

E6805 Symbolic Host Support

- Full C source level symbolic debugging
- Simplifies Motorola EVM, EVS emulation board debugging
- Companion to C6805 Code Development System

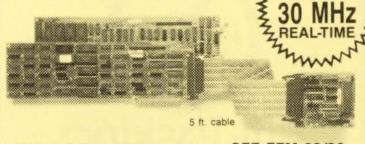
Call today! 519-888-6911

BYTE CRAFT LIMITED
421 King St., N.
Waterloo, ONT. N2J 4E4

CIRCLE NO 325

☆ ☆ ☆ ☆
"The Best 8051 Emulator"

NEW 30 MHz REAL-TIME



8051

SEE EEM 89/90 Pages D 1324-1326

PC based emulators for the 8051 family

8031, 8032, 8051, 8052, 80C152/154/321/451/452/51FA/51GB/515/517/535/537/552/562/652/851, 80532, 83C451/552/652/751/752/851, 8344, 87C451/552/751/752, 8751, 8752, DS5000 + CMOS . . . more.

- PC plug-in boards or RS-232 box.
- Up to 30 MHz real-time emulation.
- Full Source-level Debugger *w/complete* C-variable support.
- 48 bit wide, 16K deep trace, with "source line trace."
- "Bond-out" pods for 8051, 83C552, 83C451, 83C652, 83C751, 80C515/80C517, 83C752.

Prices: 32K Emulator 8031 $1790;* 4K Trace $1495*

CALL OR WRITE FOR FREE DEMO DISK!
Ask about our demo VIDEO

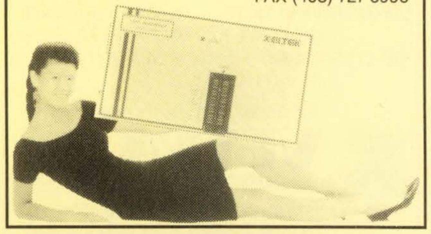
nOHau CORPORATION
51 E. Campbell Avenue
Campbell, CA 95008
FAX (408) 378-7869
(408) 866-1820

*US only

CIRCLE NO 326

UNIPRO,
the PC/XT/AT/386 based universal programmer/tester programs PROMs, EPROMs, EEPROMs, up to 4MB and 32-bit wide, PALs, PLDs, GALs, EPLDs, PEELs, and Micro Controllers. JEDEC file compatibility and Test Vector verification allow the use of most popular PLD compilers. The unit also tests TTL/CMOS Logic ICs and Dynamic/Static RAMs. 40-pin Gold ZIF socket, built-in protection for short circuit and over current, high speed parallel interface to the PC, and menu-driven software are included at $585.

XELTEK
764 San Aleso Ave
Sunnyvale, CA 94086
TEL (408) 727-6995
FAX (408) 727-6996



CIRCLE NO 327

Key Abstracts in
Software Engineering

Provides details of recently-published papers chosen from leading international journals and conference proceedings. Selection of these key abstracts is carefully monitored by INSPEC's team of qualified subject experts and brought to you each month in an easy-to-read format.

Contents Include: Programming Support, High Level Languages, Compilers and Interpreters, Operating Systems, Database Management Systems, Distributed DBMS, Relational DBMS, Software Engineering Management, Software Techniques and Other Aspects.

For more information on this title and all 20 of our Key Abstracts, please contact:

INSPEC

IEEE Service Center, INSPEC Department
445 Hoes Lane, Piscataway, NJ 08855-1331
Phone: (908) 562-5553 Fax: (908) 981-0027

1991 Subscription price – **$79.00 Member $142.00 Non-member**
All subscriptions must be prepaid and are on a calendar year basis

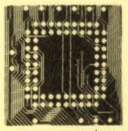
CIRCLE NO 328

***Protel Autotrax*™**

Best PCB design solution for mixed Digital, Analog, and SMT boards

Our NEW and POWERFUL *Protel Autotrax*™ is a fully integrated PCB layout system with automatic component placement and auto-routing in a single working environment. Its latest features will definitely push the price/performance of mixed technology PCB designs to the highest level, boost your design productivity, and deliver your products to the marketplace faster than your competitors.

- Integrated automatic component placement and autorouting
- On-the-fly library components creation
- 45°, 90° and curve tracks routing
- Powerful user-definable Macros
- Auto-panning
- PostScript ™ printing
- Switchable Metric/Imperial grid
- Intelligent Pad to Pad autorouting
- Automatic power/ground relief for SMD pads
- Automatic Copper Pour leaves clearance for tracks & pads



From schematic design, manual and automatic PCB design, Rip-up and Retry autorouting, to Gerber viewing and editing, we offer free tech and EMS support, 24-hour BBS and 30-day money back guarantee and our prices start at $395.

Free Evaluation Package
Toll Free: 800-544-4186

Protel Technology, Inc.
50 Airport Parkway, San Jose, CA 95110
Tel: 408-437-7771 Fax: 408-437-4913

CIRCLE NO 329

X.25	SDLC
QLLC	HDLC
ADCCP	PAD

- C source code
- ROM-able
- Full porting provided
- No OS required

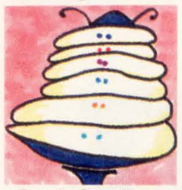
GCOM, Inc.
41 E. University
Champaign IL 61820
(217) 352-4266

Specialists in Computer Communications
FAX 217-352-2215

CIRCLE NO 330

To advertise in Product Mart, call Joanne Dorian, 212/463-6415

WANTED
LAS Bug #331154



Stack Overflow

Locate him with the deep memory of the Logic Analysis System

4k to 64k memory depth
16 level triggering with pass counter
40 to 320 channels
A complete Logic Analyzer on a Card
Interactive software under Microsoft™ Windows
Source code in C for automatic testing

BITWISE DESIGNS, INC.
701 River Street
Troy, NY 12180
(800) 367-5906
(518) 274-0755
FAX (518) 274-0764

CIRCLE NO 331

RELIABILITY AND MAINTAINABILITY PREDICTION AND FMECA ANALYSIS SOFTWARE

Hundreds have used this leading computer-aided engineering software since 1982.

Powertronic Systems offers software to predict Reliability and Maintainability and for Failure Modes Effects and Criticality Analysis. Hundreds of users have selected from PSI's large, versatile and integrated software family for military and industrial equipment and for both electrical and mechanical systems. And, these programs are either interactive or can be input from batch modes from existing CAE or database programs.
Programs implement MIL-STD-1629; MIL-HDBK-217 including E Notice 1; and MIL-HDBK-427.

PSI **Powertronic Systems, Inc.**
P.O. Box 29109 New Orleans 70189
(504) 254-0383 FAX (504) 254-0393

CIRCLE NO 332

FREE DEMO!

DspHq

- Complete DSP development for IBM-PC
- Integrate DSP hardware, data acquisition
- Open-architecture design
- Algorithm design & simulation
- Add custom & 3rd party analysis routines
- Menus / Script files / Graphics
- Easy to use & only **$495!**

BittWare Research Systems 800-848-0435
Download Demo: (3-12-24/N/8/1) 301-838-3205

CIRCLE NO 333

New from VETRA
CONVERT between PC KEYBOARD ↔ RS-232

VIP-331 Smart Pipe™— Converts RS-232 to standard PC keyboard input.
VIP-335 Smart Splice™— Accepts both RS-232 and a keyboard as inputs to PC's keyboard input.
VIP-341 Reverse Pipe™— Converts PC keyboard output to RS-232.
VIP-345 Reverse Splice™— Feeds a PC keyboard's output to both a PC and an RS-232.
VIP-411 Smart Ae™— Encodes discrete or matrix switches to RS-232.

For detailed information, please contact

VETRA SYSTEMS CORPORATION
1670 Old Country Rd., Plainview, NY 11803
Tel (516) 454-6469 • FAX (516) 454-1648

CIRCLE NO 334

8086 8051 8096 ROYALTIES 68000 68HC11 Z80

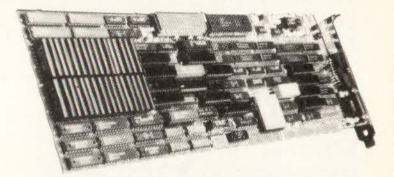
Multi-Tasking EXECS

U S Software offers hi-performance software tools for embedded applications.

Get the full details by calling:
800-356-7097
503-641-8446
503-644-2413 (FAX)

U S SOFTWARE
United States Software Corporation
14215 NW Science Park Drive
Portland, Oregon 97229

CIRCLE NO 335



4MEG VIDEO Model 10
Flexible Image Processor and Application Accelerator For The PC/AT

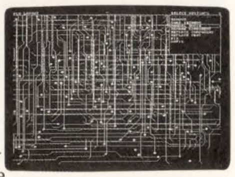
- 8 to 8000 Pixels per Line
- 2 to 19 MHz sampling/display rate
- 10 MIPs Programmable Accelerator
- 4 Megabytes of Reconfigurable Image Memory
- RS-170, RS-330, and CCIR input/output
- Variable timing for nonstandard formats
- Genlock to external timing sources
- Analog or digital inputs
- Software programmable timing/resolution

EPIX
310 Anthony Trail, Northbrook, IL 60062
708-498-4002 FAX: 708-498-4321

CIRCLE NO 336

FREE MAXI/PC DEMO DISK

$995 PCB CAD SOFTWARE



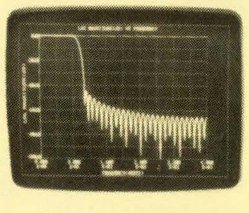
MAXI/PC integrated software includes schematic capture, layout, autorouting, and outputs. 30 day, money back guarantee.
For your free demo disk and brochure, call today or circle the reader service number.

1-508-692-4900

MAXI/PC *PCB CAD SOFTWARE*
RACAL RACAL-REDAC
238 Littleton Road, P.O. Box 365
Westford, MA 01886-9984

CIRCLE NO 337

DIGITAL FILTER DESIGN SOFTWARE
SUN 4, IBM PC, MACINTOSH
IIR, FIR, PARKS-MCCLELLAN
COEFFICIENT QUANTIZATION
TRANSFER FUNCTION ANALYSIS
Code Generators • Hardware Development Tools • Demos Available



MOMENTUM DATA SYSTEMS
1520 Nutmeg Place #108, Costa Mesa, CA 92626
TEL: (714) 557-6885, FAX: (714) 557-6969

CIRCLE NO 338

30-day money-back guarantee!

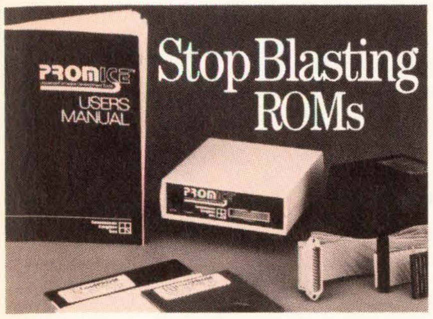
TESS® PC Block Diagram Simulation

Modems, DSP, satcom, nav, controls...
Times are tough: shorter schedules, more complexity. TESS can help you:
- Detect gotchas before breadboarding.
- Find the best design alternatives fast.
- See the effects: filters on signals & loops, ϕ-noise, multipath & adjacent channels.
- Simulate mixed-signal ASICs & systems.
- Put a lab on your desk for just $695!

Demo disk: 404-751-9785 Fax: 404-664-5817
TESOFT Box 305, Roswell, GA 30077

CIRCLE NO 339

To advertise in Product Mart, call Joanne Dorian, 212/463-6415



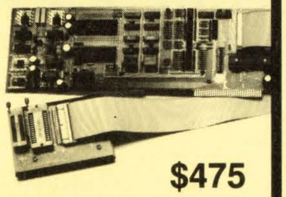
PROMICE emulates 8 bit ROMs from 2716-27080, or 16 bit ROMs 27C1024 or 27C2048. (Inquire about emulating other ROMs. Non-JEDEC ROMs require custom cable.) ■ Sophisticated LoadICE™ Host Software downloads, uploads and edits ROM contents, supports MS-DOS, UNIX, MAC & VMS. Software sources are included. ■ Bi-directional **Serial** link, autobaud to 57.6KB–loads 1 Mbit in 25 secs. ■ Bi-directional **Parallel** port (option)–loads 1 Mbit in 4 sec. ■ Emulate up to 2 ROMs per unit, daisy-chain up to 256 ROMs from one port! ■ **New! Analysis Interface**™ (option) implements a ROM-based UART for sophisticated debugging.

Grammar Engine Inc
1161 Cherry Street
San Carlos
California 94070
415/595-2252

CIRCLE NO 340

UNIVERSAL PROGRAMMER

PAL
GAL
EPROM
EEPROM
PROM
87C51...
874x...



$475

5ns PALs 4 Meg EPROMs
26V12 & 22V10 Gals
FREE software updates on BBS

GANG PROGRAMMER
- 4 32pin Sockets (8 Socket option)
- 2716-27010 EPROMs

$215

Call - (201) 994-6669
Link Computer Graphics, Inc.
4 Sparrow Dr., Livingston, NJ 07039 FAX:994-0730

CIRCLE NO 341

```
#include <stdio.h>

typedef struct {
   int foo;
   double bar;
   char baz[36];
} quux;

/***     foobar - inspect baz in a quux
*/

static quux * foobar(bar, quuxptr)
   int bar;
   quux * quuxptr;
{
   int i;

   for (i = bar; quuxptr->baz[i]; i++)
      if (quuxptr->baz[i] == '\n')
         return (quuxptr);

   return (NULL);
}
-- More --_
```

6 800 - Family Development Software

Combine our software and your editor for a powerful development system. Our C-Compilers feature a complete implementation (excluding bit fields) of the language as described by Kernigan & Ritchie and yields 30-70% shorter code than other compilers. Our Motorola-compatible Assemblers feature macros and conditional assembly. Linker and Terminal Emulator are included.

Wintek Corporation
1801 South St., Lafayette, IN 47904
(800) 742-6809 or (317) 742-8428

CIRCLE NO 342

Elegant, concise, fast & standardized
FLOATING POINT
libraries for embedded applications

Based on the IEEE 754 standard, FPAC (32 bit) and DPAC (64 bit) libraries are mature, well documented, and fully tested. The libraries are fully ROMable and include the following:

- Basic Operations
- Square Root
- Trigonometric
- ASCII Conversion
- Integer Conversion
- Logarithmic

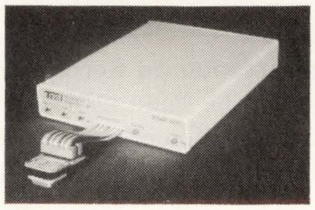
U S Software supports most Intel, Motorola, Zilog and Hitachi micros, including 80X86, 80386, 680X0, 80960, 8051, 8096, 68HC11, Z80, 6809 and 6301.

For additional information, please contact:

U S SOFTWARE
United States Software Corporation
14215 NW Science Park Drive
Portland, Oregon 97229
800-356-7097
503-641-8446
503-644-2413 (FAX)

CIRCLE NO 343

IN CIRCUIT EMULATORS
68HCO5 AND 68HC11



The TECICE-HC05 and TECICE-HC11 are low cost real time in circuit emulators for the Motorola 68HC05 and 68HC11 families of single chip microcomputers. Any host computer with a serial port and terminal emulation software can be used with these emulators. Complete development system software is included for MS-DOS based computers. Other development systems are available for 6805, HC05 and HC11. **Complete HC05 and HC11 emulators start at $995.**

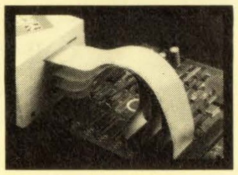
TECI
The Engineers Collaborative Inc.

Rt. #3, Box 8C
Barton, VT, USA 05822

Phone (802) 525-3458
FAX (802) 525-3451

CIRCLE NO 344

ROM-IT



EPROM EMULATION SYSTEM

- Emulates up to 8 1-Megabit EPROMS with one control card.
- Downloads 1-Megabit programs in less than 10 seconds.
- Allows examination and modification of individual bytes or blocks.
- Accepts Intel Hex, Motorola S-Record, and Binary files.
- Software available for IBM PC and Macintosh systems.

Call or fax today for more information!

Base 27256 EPROM System **$395.00**
Other configurations available.

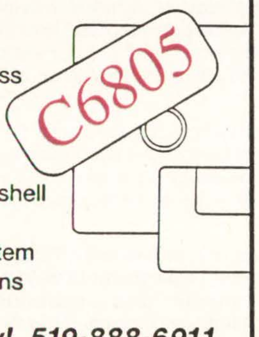
Incredible Technologies, Inc.
709 West Algonquin Road
Arlington Heights, Illinois 60005
(708) 437-2433 Fax (708) 437-2473

Visa, MasterCard, and American Express accepted.

CIRCLE NO 345

C6805
CODE DEVELOPMENT SYSTEM

- First 6805 C compiler
- Built-in cross assembler
- Includes integrated shell
- Expert system optimizations

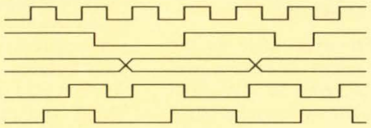


Call today! 519-888-6911

BYTE CRAFT LIMITED
421 King St., N.
Waterloo, ONT. N2J 4E4

CIRCLE NO 346

Low Cost Logic Simulator



DLsim™ Digital Logic Simulator

- Event driven, nine state functional and timing simulation
- 16,000 gate capability without additional memory
- Direct support from JEDEC files for PLDs and GALs
- Compatible with SCHEMA or ORCAD schematic files
- Runs on IBM PC/XT/AT or compatibles
- Interactive logic viewer (EGA/VGA or HERCULES)
- Supports HP Laser or EPSON dot matrix printers
- Includes TTL, ALS and CMOS libraries with source
- No copy protection
- **Complete package only $495**

CADsim Technologies

525 Melbourne Ct., Newbury Park, CA 91320
(805) 499-8653

All trademarks belong to their respective owners

CIRCLE NO 347

simple multitasking executive
smx *v2.0*

for REAL-TIME EMBEDDED SYSTEMS

new features:

- ❑ Full MS-DOS compatibility
- ❑ 80x87 & math emulator support for reentrancy
- ❑ Stack size up to 64K per task
- ❑ Protected heaps & task-reentrant calls
- ❑ v25 register bank support
- ❑ PC demo source code
- ❑ Fast pipe macros

Dev. Kit & No Royalty License $1995
Source Code - $1000
Evaluation Kit & PC demo - $95
User's Guide & demo disk - $25

- ❑ 80x86 family
- ❑ ROM'able
- ❑ preemptive scheduler
- ❑ libraries for all memory models
- ❑ works with Microsoft®C and Turbo®C, assembler and debugger
- ❑ 70 microsecond task switch (10MHz 80188)
- ❑ 15 microsecond interrupt latency
- ❑ 5KB to 25KB code size
- ❑ 1 year free support & updates
- ❑ 30 day money-back guarantee

MICRO DIGITAL
Cypress, CA 90630-5630
1-800-366-2491
FAX 714-895-2164

CIRCLE NO 348

To advertise in Product Mart, call Joanne Dorian, 212/463-6415

CAREER OPPORTUNITIES

1990 Recruitment Editorial Calendar

Issue	Issue Date	Ad Deadline	Editorial Emphasis
Magazine Edition	Oct. 1	Sept. 10	Computer Boards, Analog ICs, Digital ICs, Test & Measurement
News Edition	Oct. 4	Sept. 14	ICs/LAN Chips/Microprocessors, AI/Expert Systems, Special Supplement: Instruments
Magazine Edition	Oct. 11	Sept. 20	Analog ICs, Computer-Aided Engineering, DSP IC Directory, Displays, International Technology Update
News Edition	Oct. 18	Sept. 28	CAE/Hardware, Datacom, Regional Profile: Idaho, Colorado, Utah
Magazine Edition	Oct. 25	Oct. 4	Test & Measurement Special Issue—Digital Instruments, Computers & Peripherals, ICs & Semiconductors, System Software
Magazine Edition	Nov. 8	Oct. 18	Signal Processing, Computer-Aided Engineering, Computers & Peripherals, Software, Wescon Show Issue
News Edition	Nov. 15	Oct. 26	Displays, Defense, Special Supplement: Interconnect
Magazine Edition	Nov. 22	Nov. 1	17th Annual Microprocessor Directory, ICs & Semiconductors, Test & Measurement, Workstations
News Edition	Nov. 29	Nov. 8	ICs/Communication Controllers/Microprocessors, DSP, Regional Profile: Illinois, Minnesota & Michigan

Call today for information on Recruitment Advertising:
East Coast: Janet O. Penn (201) 228-8610
West Coast: Nancy Olbers (603) 436-7565
National: Roberta Renard (201) 228-8602

A GREAT CAREER MOVE.

Cleveland...where a revitalized downtown and billions of dollars in new construction have energized the entire region. Where clean lakes, beautiful parks, major league sports teams, world class museums, and an exciting nightlife offer variety and relaxation. Where you can still take advantage of the lowest average housing costs of any major city in the U.S. And where you'll find the Industrial Computer and Communication Division of Allen-Bradley, the global leader in industrial controls and automation, now making a move to the forefront of software engineering for Computer Integrated Manufacturing. To enhance our efforts, exceptional opportunities are now being offered to enterprising, innovative professionals.

SENIOR HARDWARE DESIGN ENGINEER: responsible for the definition, design, implementation, test and release of system level hardware for the DCIO platform. BSEE or equivalent, and a minimum three years' experience in the design of digital hardware for industrial control systems required; MSEE preferred.

SYSTEM PROGRAM MANAGER: oversees system test plans and implementation, software quality, and system engineering. BSEE or BSCS, and experience with software project management, system test planning and execution, and system engineering essential; graduate degree desired.

PROJECT ENGINEER: responsible for the development of interfaces for reusable software components, and specification of key elements in the architecture. A major emphasis is placed on data management services design. MSCS or equivalent, and experience in software system design, database design, and real time factory control required.

QUALITY ASSURANCE ENGINEER: will monitor projects/products throughout their life-cycle to ensure reliability, application functionality, and safety. Must possess a BSEE, and a minimum two-five years' experience in product assurance and/or quality engineering/test engineering with analog and digital, microprocessor-based products. Knowledge of QA techniques and statistics a must; MSEE or MBA desired.

SENIOR SUPPLIER QUALITY ASSURANCE ENGINEER: relied upon to organize, plan, and monitor the receiving inspection and supplier selection and evaluation functions; and assists in the implementation and maintenance of TQMS and Customer Satisfaction programs. Requirements include a minimum five years' technical quality discipline experience and familiarity with SPC/Process Control, weld process (AWS-D1.1), paint process/testing, and military specifications. A BS degree and ASQC certification preferred.

You'll enjoy the technical challenges, the opportunity for growth, and the rewards of excellent compensation including relocation assistance offered by Allen-Bradley. So if you're looking for a better quality of life for you and your family, now's the time to make a great career move.

For more details, send your resume with salary history to: **P.J. Liedtke; Human Resources, Department EDN-917; Allen-Bradley; Industrial Computer and Communication Group; 747 Alpha Drive; Highland Heights, Ohio 44143.** An equal opportunity employer, M/F/H/V.

A-B QUALITY **ALLEN-BRADLEY**
A ROCKWELL INTERNATIONAL COMPANY

OUTSTANDING

In the world of electronic defense systems, being good isn't good enough. It takes a company-wide commitment and the very best professionals to put technology on target.

At E-Systems Garland Division, we understand that to succeed in the complex world of electronic defense, we've got to be better than good. We've got to go one step beyond–to redefine the meaning of excellence. That's because there are no near-misses in our business. Only bulls-eyes. And bulls-eyes on bulls-eyes.

So it's no accident that we're one of America's fastest growing suppliers of electronic defense systems. Outstanding systems, products and technologies come from superlative professionals.

E-Systems Garland Division is a Dallas-based leader in advanced signal and image processing technologies, sophisticated receiver systems, and systems integration for programs like JSIPS and the Distributed Wargaming System.

Be outstanding in your field.

If you're a technical professional, we invite you to stand out with E-Systems Garland Division.

Digital Signal Processing

- Ada and Oracle Software Development
- 68000 Board Level Design

Digital Image Processing

- Digital/Analog Design
- UNIX/"C" Software Development
- VHSIC, VLSI, MMIC, GaAs Design
- 68000 Real-Time Firmware Design

Communications

- 68000/80xxx Board Level Design

Mass Storage

- Hi-Speed Computer Interfaces
- Systems and Product Engineering
- UNIX/"C" Software Development
- Convex Systems Software

Management Information Systems

- IDMS/Cullinet Applications Software

Take aim at the future.

Our technical careers are on target, as well; we have one of the lowest turnover rates of any company in our industry. As a technical professional with E-Systems Garland Division, you'll enjoy a superb compensation package–featuring a flexible program that lets you tailor your own benefits. And our ESOP program makes every E-Teamer part owner of the company.

Be better than good. Join E-Systems Garland Division today. Send your resume to: Ann Olson, Director of Staffing, E-Systems, Inc., Garland Division, P.O. Box 660023, Dept. 31FSM, Dallas, Texas 75266-0023.

E-SYSTEMS
The science of systems.

U.S. Citizenship Required. An Equal Opportunity Employer.

CIRCLE NO. 30

FILENET IS #1

FileNet Corporation, the pioneer in document image processing, is looking for qualified people to help fuel its rapid growth in 1990. **FileNet,** rated #1 in overall customer satisfaction in the 1990 AIIM/Datapro survey, manufactures and markets an optical disk-based system for automating the processing of paper. Used by major corporations and governments throughout the world, **FileNet** systems have become synonymous with image processing.

If you're the type of person who wants to become part of a winning team in an explosive industry, then check out the exciting opportunities **FileNet** provides. Currently, we have immediate openings available for the following positions. Qualified candidates will have 2-5 years related experience with a BS or MS in Computer Science or equivalent.

SYSTEM INTEGRATION ENGINEERS
(Dept. SIE)

- Software integration
- Development testing
- Testing tools, scripts and internal support of integration and development systems
- Must have UNIX® /"C" experience and familiarity with PC-based applications

SOFTWARE SUSTAINING ENGINEERS
(Dept. SSE)

- Software maintenance of **FileNet** product field releases
- Requires operating system expertise in UNIX®
- Analyze technical problems and implement effective solutions
- Strong background in "C", UNIX® kernel development and maintenance
- Familiarity with device drivers is a plus

SOFTWARE QUALITY ASSURANCE ENGINEERS
(Dept. SQAE)

- Writing test plans and test scripts
- Testing and validation of interactive user-oriented applications software
- Minimum 5 years experience in large computer systems validation & testing
- Knowledge of relational data base tools. Oracle preferred
- Hands on technical experience required

SOFTWARE ENGINEERS
(Dept. SE)

- PC and SUN workstation development utilizing OS/2 and Windows (X-MS-PM-Sunview)
- Applications development on workstations involving user interfaces
- Excellent verbal and written communications skills
- Proficiency in "C" and related languages

OPERATING SYSTEMS ENGINEERS
(Dept. OSE)

- We seek Senior Designers/Developers/Architects to port and customize System V release 4
- Require UNIX® kernel internals experience and a strong technical foundation to optimize performance issues, utilizing complex OS concepts.

UNIX® SUPPORT ENGINEERS
(Dept. USE)

- Third level customer support with extensive software development and/or engineering customer support experience
- Proficient in UNIX® and "C"
- UNIX® (kernel) (internal)
- Familiarity with Ethernet communication protocols

CUSTOMER SOFTWARE SUPPORT ENGINEERS
(Dept. CSSE)

- Customer problem resolution
- Installation/Configuration
- Software systems support
- Database structures
- Operating systems and concepts
- Experience in corporate software post-sales technical support to customers
- Technical degree and a minimum of 5 years strong technical experience

FileNet offers an excellent compensation package, and a challenging work environment designed to keep us THE LEADER IN THE INDUSTRY. For immediate consideration, send your resume with salary requirements to: **FileNet Corporation, Attn: Human Resources, (Put Dept. Code Here), 3565 Harbor Blvd., Costa Mesa, CA 92626.** Equal Opportunity Employer.

FileNet®
First In Image Processing.

SOMETHING GREAT IS HAPPENING...

Who we are:	At GE Medical Systems there is a unique spirit. It is the feeling that comes when you are a true global leader and innovator in state-of-the-art medical diagnostic systems. Our people are part of that spirit. Smart. Proud. Thinkers and Doers working with an elite group in the evolution of diagnostic imaging systems.
Who we need:	**SOFTWARE SYSTEMS INTEGRATOR** Develop and execute a strategy for integrating software components of a product development program. **REQUIRES:** BS/MS EE, CS and real-time software engineering/large scale systems integration experience. **SOFTWARE DEVELOPMENT ENGINEER** Develop X-Ray product applications/diagnostics/calibration software and participate in cross functional designs/reviews. **REQUIRES:** BS/MS EE, CS and structured software development, PL/M86 and INTEL microprocessor experience, digital hardware knowledge. **PROJECT ENGINEER-EMI** Develop designs/design guidelines for diagnostic imaging systems to meet international regulatory requirements for EMI/EMC; define project tasks and schedules; track progress. **REQUIRES:** BS/MS EE, ME and EMI, electronic-packaging/power grounding/cabling design experience, international regulatory requirements knowledge. **SOFTWARE SYSTEMS ENGINEER** Specify and develop software for applications/data acquisition/display or diagnostics. **REQUIRES:** BS/MS EE, CS and real-time/scientific software design experience, including "C" & UNIX. **SENIOR SOFTWARE ENGINEER** Lead the development and implementation of system level requirements for the control of patient handling apparatus. **REQUIRES:** MS EE, CS and 7+ years experience in the development of software for closed loop servo controls.
How to reply	GE's highly competitive salary and benefits befit an industry leader. Please send resume in strict confidence to: **JMB, GE Medical Systems, P.O. Box 414, W-407, Milwaukee, WI 53201.** Replies will be made, within 20 days, to candidates of interest.

GE Medical Systems

An Equal Opportunity Employer

TECHNOLOGY AS DIVERSE AND EXCITING AS OUR COLORADO ROCKY MOUNTAIN WEEKENDS!

As you explore your career alternatives, you'll discover few companies that offer the technological diversity, intensity and excitement found at Martin Marietta Astronautics Group.

Superior research capabilities and a sure grip on leading edge technologies support Astronautics Group's large, diverse contract base. Spacecraft fabrication, instrumentation, launch vehicle systems, propellant and power management, electronic systems and software, guidance and control, large space structures, robotics and artificial intelligence are the foundation for an awesome breadth of career opportunities.

In the course of a career at Martin Marietta Astronautics Group, you can expect to work on a broad, exciting repertoire of projects, often across several of the Group's operational companies. This philosophy creates exceptional opportunities to learn, advance and succeed.

The Colorado Rocky Mountain Good Life!

If you like the outdoors, you'll love it here. Coloradans enjoy up to 300 days of sunshine yearly, an average temperature of 64 degrees, and plenty of opportunities to take advantage of it all. Such as golfing, sailing, rafting, camping and fishing in the high-country summers.

For winter sports enthusiasts, the light powder snow of Colorado's world-class ski areas is just a few hours' drive from the excitement of the Denver metroplex. And not only is Colorado one of the most enviable places to live, it is also one of the most affordable.

What kind of individuals thrive at Martin Marietta Astronautics Group? Those with a hunger to learn more and do more. Highly professional people who believe in teamwork, yet understand the importance of individual accomplishment. If you're among them and possess an appropriate technical degree, explore career opportunities in software, systems, testing, propulsion, integration, materials, structures, dynamics, mission analysis, logistics, RF systems, guidance and control, quality control and manufacturing.

Please send your résumé to Martin Marietta Astronautics Group, Human Resources Department P00077, P.O. Box 179, DC1311, Denver, CO 80201. Many positions require U.S. citizenship. Martin Marietta is an equal opportunity/affirmative action employer, M/F/H/V.

MASTERMINDING TOMORROW'S TECHNOLOGIES

MARTIN MARIETTA

Timing is everything.

Real-time applications require precision like no others — because responses must be split-second and right the first time. At Ready Systems, we design the comprehensive software tools allowing real-time designers to create the ultimate in embedded systems. Aids like VRTX™ — the real-time operating system standard, ARTX™ — the Ada programming version and CARDTools™.

With the market growing at 55% a year. our timing has never been better. And if you're looking for real career challenge, neither has yours. Join us in one of the following positions:

Software Development Engineers

Work on user interfaces, operating system internals, or Ada run-time systems. Requires experience with UNIX®, C, C++, Ada and real-time applications.

Software Applications Engineers

Provide post-sales support, design product demos, train customers and provide application-specific consulting. You'll need a strong background in application programming in C, Ada and/or Assembler. Familiarity with embedded microprocessor-based systems is also required.

Product Marketing Manager

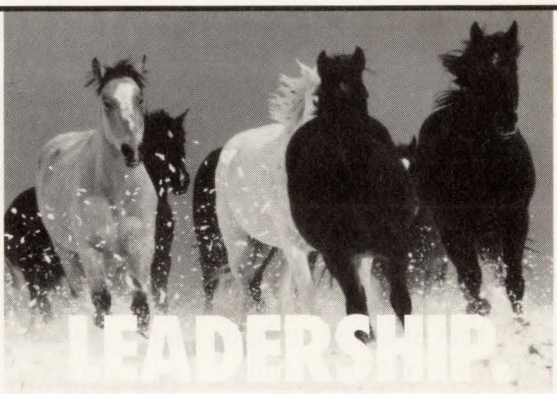
You'll lead the charge for our industry leading VRTX Velocity and VRTX32/68K products. You'll need a BSEE or equivalent, 2+ years' high tech marketing experience, and exposure to UNIX, VAX/VMS and RTOS.

A Ready Reply.

Please send your resume and salary history to: Ready Systems, Professional Staffing, Dept. EDN, 470 Potrero Avenue, Sunnyvale, CA 94086. EOE/AA.

Trademarks are registered to their respective companies.

READY SYSTEMS



The capacity to show the way by taking the lead. To influence or direct the activities of others.

Some appear to be the leader. But actions speak much louder than muscle. We believe in the personal power of the individual. Which is why, at Microprocessor Semiconductor Products Sector, we encourage our people to be champions. To establish goals. To influence by example. As a result, we're an innovator in the semiconductor industry. Openings are now available for Marketing Professionals in the 88000 RISC operation with expertise in the following areas:

Software Engineering Manager

Lead a technical team in porting and developing 68000 and 88000 C compilation systems for the latest UNIX System V technology. Individual must have prior experience managing a software team with the purpose of producing a compilation system. Minimum requirements are BSCS/MSCS and six years of related experience.

Software Engineering Manager

Lead a technical team in developing optimizing C and FORTRAN compilers for state-of-the-art microprocessors. Individual must have experience in computer architecture, compiler code generation, and compiler optimization techniques. Minimum requirements are MSCS and five years of related experience.

Software Engineers — Floating Point

Become an expert on 68000 and 88000 floating point software. Working closely with chip designers, design and implement a floating point software package for each family. Minimum requirements are BSCS or BSEE plus three years of experience in industry.

Software Engineers — C & Fortran Compilers

Participate in the development of C and FORTRAN compilers for Motorola's leading edge microprocessors. Several positions are open for individuals experienced in ANSI C and/or FORTRAN77 compiler front ends and highly optimized compilers. Knowledge of computer architecture, compiler code generation, and compiler optimization techniques are highly desirable. Minimum requirements are MSCS plus three years of related experience.

Embedded Control Strategic Marketing

Develop and communicate marketing programs and strategies for embedded control/real-time applications with the 88000 family of products. A key focus will be on facilitating design wins in targeted areas. Includes development and execution of customer presentations, training programs and promotional programs. Must have 2+ years experience in the embedded control market.

Computer Strategic Marketing

Develop and communicate marketing programs and strategies for the computer market (desktop to large scale parallel processor systems). Major emphasis on obtaining design-wins. Requires 2+ years experience in computer systems and UNIX marketing or planning.

Applications Engineer

High-end Microprocessor/Microcontroller Applications Engineer. Work includes customer telephone support, application note preparation, article writing, and customer presentations. Applicable devices include M68000, 88000, HCII, 352 families. BSEE and 3+ years experience.

Please submit resume to: **Motorola Semiconductor Products Sector, Dept. ATX-185, 505 Barton Springs Rd., One Texas Center, Suite 400, Austin, TX 78704. 24 hour FAX (512) 322-8811.** An Equal Opportunity/Affirmative Action Employer.

MOTOROLA

Semiconductor Products Sector

IT'S IN OUR NATURE.

ENGINEERING

COMPUTER SOFTWARE ENGINEERS

Lockheed Missiles & Space Company is leading the way in the development of advanced spacecraft and other software systems. We're seeking highly qualified professionals in fields such as:

- **Ada, C, FORTRAN Software Development**
- **Software Design**
- **Software Test/Verification**
- **UNIX® Internals**
- **Real-time Control Software**
- **CASE Tools**
- **Software Systems Integration**
- **Software Systems Engineering**
- **Networking**
- **Mission Planning**
- **Image Processing**
- **System Architecture**
- **Relational Databases**

All positions require at least a BS degree and 3 years' experience. In addition to challenging projects, we offer an excellent compensation and benefits package.

For consideration, please send your resume to Lois Moulton, Professional Staffing, Lockheed Missiles & Space Company, Dept. 517NKLM, Sunnyvale, CA 94088-3504. U.S. citizenship is required.

Candidates possessing a current EBI/SBI preferred. All candidates should be able to pass an extensive background investigation. We are proud to be an equal opportunity, affirmative action employer.

UNIX is a registered trademark of AT&T.

Lockheed Missiles & Space Company

INNOVATION: giving shape to imagination.

ADVERTISERS INDEX

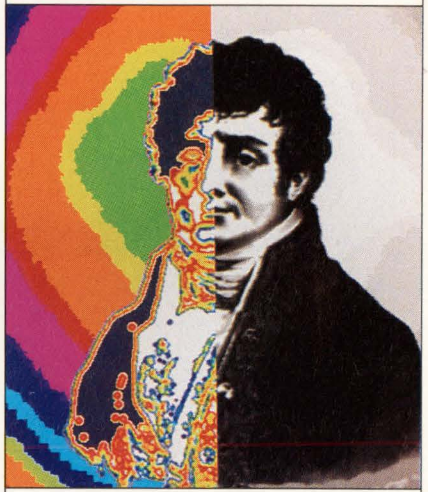
American Automation	6
Applied Microsystems Corp	20
Ariel	34
Bittware	64
Bitwise Designs Inc	64
Bytecraft	63, 65
Cadre Technologies	4-5
CADSim Tech	65
Digital Equipment Corp	36-37
EPIX Inc	64
Eyring Research Institute	35
Forth Inc	C3
GCOM Inc	63
Grammer Engine	65
Huntsville Microsystems Inc	40
IEEE	63
Incredible Tech	65
Integrated Silicon Systems	51
Link Computer Graphics Inc	65
Logical Devices Inc	28
Micro Digital	65
Microtek	2
Momentum Data Systems	64
National Instruments	39
NEC Electronics Inc	C2
Pactec Corp	38
Powertronic	64
Prem Magnetics Inc	C4
Protel Tech Inc	63
Racal-Redac	64
Samsung Semiconductor	8-9
Simucad	72
Software Component Group	17
Spectral Innovations	71
TECI	65
Tesoft	64
Transera	1
US Software	64, 65
Vetra Systems	64
Westcor	19
Wintek Corp	65
Xeltek	63

Recruitment Advertising 66-71

Allen-Bradley
E-Systems Garland Div
Filenet
Martin Marietta
Motorola SPS
Ready Systems
Seiscor Technologies

This index is provided as an additional service. The publisher does not assume any liability for errors or omissions.

64 MHz Array Processing On Your Mac.


"Is this what you had in mind, Mr. Fourier?"

With a 64 MHz floating-point coprocessor that can handle a 1K FFT in just 1.7 mSec, MacDSP is fast enough for real-time applications.

Over 150 functions for DSP, Math and analog I/O help you get up to speed faster in familiar languages like C, Pascal, BASIC, and FORTRAN.

Ideal for signal processing, image processing, and array processing. Call or write for a free Hypercard demo disk.

Spectral Innovations, Inc.
4633 Old Ironsides Dr., Ste. 450,
Santa Clara, CA 95054
408-727-1314

CIRCLE NO. 18

GOOD ANALYSIS DOESN'T HAVE TO COST A FORTUNE.

Deep-seated quirks hidden within your complex designs. They can drive you crazy. Delay product time-to-market. And strain your design budget. But there is a solution.

And Simucad offers it now. In fact, in1984 we wrote the textbook on cost-effective design analysis, with our industry-standard SILOS simulation software.

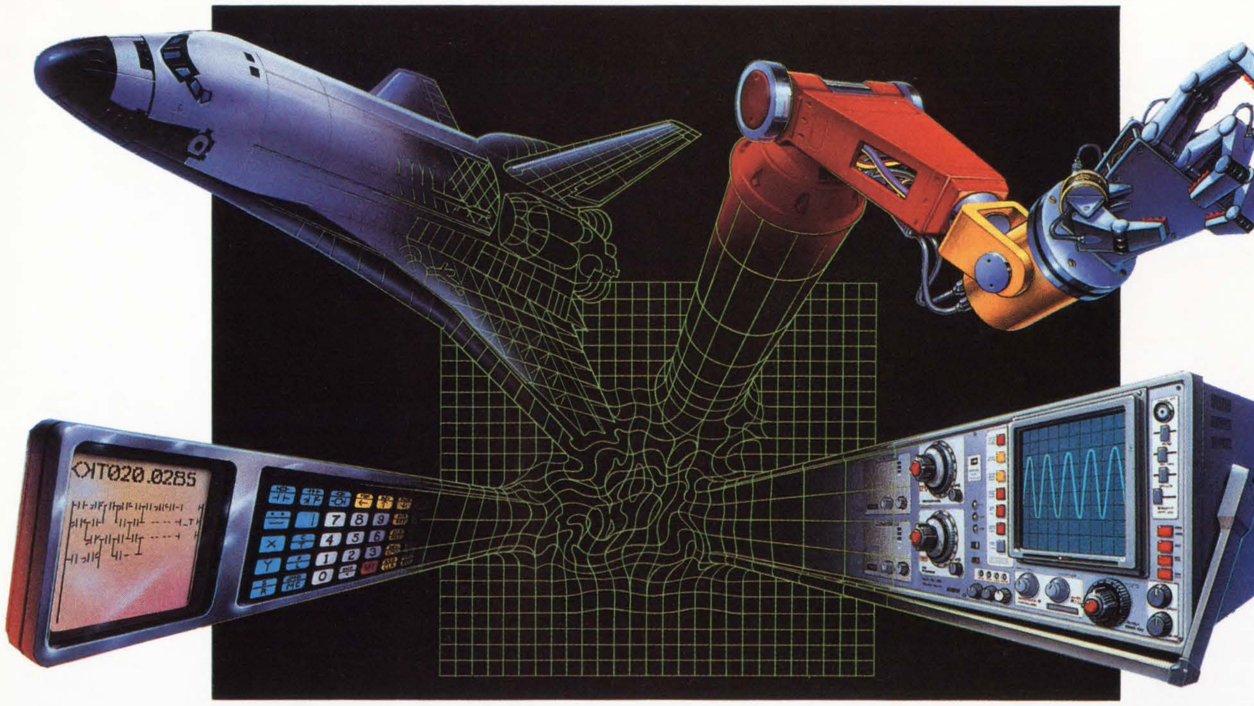
Now, the leader is back. With SILOS II, the world's most powerful interactive logic and fault simulation system. Analyze your subject, top-down and bottom-up, for less than you'd expect. SILOS II spells hardware independence. And the ability to address every element of your design's complex personality–from circuit description to final verification.

Get a head start on your competition. With SILOS II, the essential analysis software. It'll improve both your mental and fiscal health. Call us today,(415) 487-9700.

SIMUCAD

Simucad Inc., 32970 Alvarado-Niles Rd., Union City, CA 94587

CIRCLE NO. 19



High Performance Programming Solutions

When you need to accelerate your productivity, go with the proven performance software. Forth. Because with Forth your concept will become reality faster. Just listen to what people are saying about how Forth cuts programming times and helps them consistently meet or beat deadlines:

"We have found that the use of Forth has cut our development time by 90% for equivalent assembly language and 50% for equivalent high-level language coding." Jerry Tifft, **I&CS**, *May 1989*

"The development speed was fabulous." Ray Vandewalker, **Embedded Systems Programming**, *April 1990*

"Forth gets high marks for transportability, testability, and programming productivity. Until you use it, you can't believe how quickly a rough-cut version of the problem can be up and running." Lawrence L. Cone, **BYTE**, *October 1985*

"The speed that you can achieve in writing quick pieces of code is ten to a hundred times that of a traditional compiled language." Byron Palmer, **Computers in Physics,** *Mar/Apr 1988*

"Forth puts the computer power directly in the programmer's hands. . . . Based on experience with similar systems and development, the programmers estimated it would take 10 times longer to develop . . . in another language. With Forth, the . . . prototype was ready for use in six weeks." Cameron Lowe, **Telephony**, *December 1987*

"Forth is interpretive and highly interactive, giving developers the ability to prototype applications easily. . . . The major Forth supplier is FORTH, Inc. . . . " Ray Weiss, **Electronic Engineering Times**, *August 6, 1990*

Forth will deliver for you, too. Especially polyFORTH, a superset of the Forth programming language, from FORTH, Inc. The best way to cut your development time on real-time applications such as factory automation and embedded systems.

With polyFORTH you get a multiuser, multitasking, real-time operating system. An editor, assembler, compiler, debugger, hundreds of library routines, and a wealth of utilities. All fully integrated into an easy-to-use, resident development environment.

To learn more about how polyFORTH can accelerate *your* programming, call FORTH, Inc. The high performance solutions people.

(800) 55-FORTH

FORTH, Inc.
111 N. Sepulveda Blvd.
Manhattan Beach
California 90266-6847
(213) 372-8493
FAX (213) 318-7130

CIRCLE NO. 20

We Sing in the Shower!

At Prem we realize that into every customer's life, a little rain must fall. That's why we've always given our entire line of Printed Circuit Power Transformers a protective "raincoat" (inside and out) of 100% solids epoxy resin that withstands most aqueous and solvent printed circuit board cleaning systems. Pins are then retinned after impregnation to ensure good solder contact on the P.C. boards. **Prem's Printed Circuit Power Transformers are manufactured to withstand the environments of commercial and industrial use.**

PREM*IER* SERVICE
Prem*ier* Service means that we gladly ship small quantities immediately. Depending on the size of your order, future shipments can be scheduled for up to 12 months from your order date.

CALL FOR OUR CATALOG
For your convenience, our catalog includes cross references to Microtran, Signal, Stancor and Triad part numbers.

Send for our Free Catalog today!

PREM®
MAGNETICS, INCORPORATED

3521 North Chapel Hill Road
McHenry, IL 60050
Tel. 815-385-2700
FAX: 815-385-8578

Where Quality Really Counts!

CIRCLE NO. 21