A CAHNERS PUBLICATION JUNE 20, 1991

EDN®

Special Software Supplement — TO EDN —

ELECTRONIC TECHNOLOGY FOR ENGINEERS AND ENGINEERING MANAGERS

Special Report: Draw your programs with diagram compilers

pg 12

are part of a close-knit neighborhood of Intel SuperScalar i960 microprocessors. So you get software compatibility across the board as well as an easy performance path up to 100 MIPS.

And while great price/performance and compatibility are important, they're not the only reasons some very important companies have already moved into the i960 line. They were also impressed with the comprehensive array of development tools and the outstanding technical support that made them feel right at home with the technology.

So when you are ready to move into the i960 SA/SB line, call 800-548-4725 and ask for the 960 Welcome Guide. We'll not only make the move less expensive, we'll even help you set up.

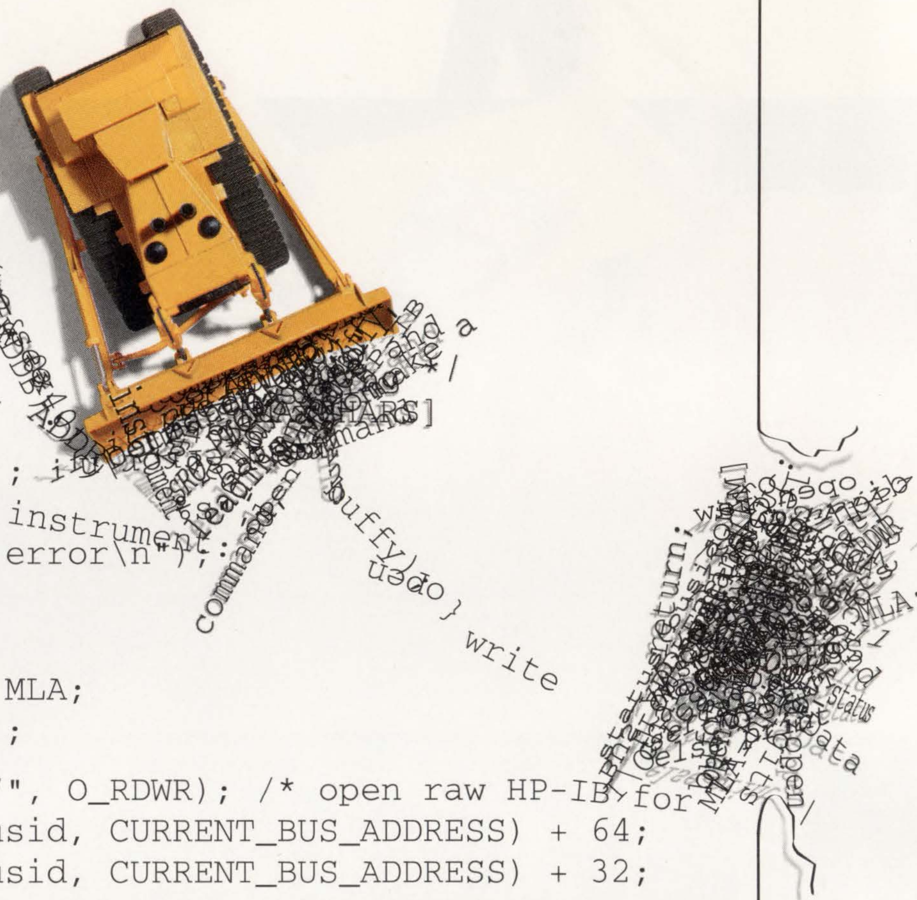
intel®

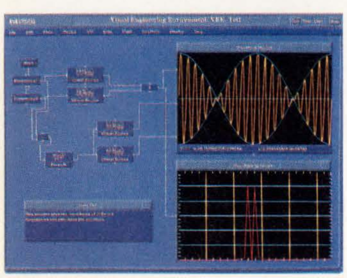
The Computer Inside.™

CIRCLE NO. 3

Finally, engineering software that clears the way to problem solving without programming.

```
void servic
int eid;
{ int stat, byte;
/*serial poll
byte=hpib_spoll(
if ( (byte<0)||! (b
    printf("SRQ Prob
   return; }
stat=my_read(eid, DVM_
if (stat>0) {
    buffy[stat] = '\0';
    printf("Data from instrument
else printf("I/O read error\n");
return; }
main() {
int busid, stat, MTA, MLA;
char command[MAXCHARS];
busid=open("/dev/hpib7", O_RDWR); /* open raw HP-IB for
MTA=hpib_bus_status(busid, CURRENT_BUS_ADDRESS) + 64;
MLA=hpib_bus_status(busid, CURRENT_BUS_ADDRESS) + 32;
stat = BUTTON_BIT ;
sprintf(command, "KM%02o", stat); /* 2 octal digits */
```





With HP VEE, you simply link the icons.

Computers are great for problem solving, if only programming didn't get in the way and slow you down. And now, it doesn't have to. Because the HP visual engineering environment (HP VEE) lets you solve problems without programming.

With HP VEE, you explore solutions visually by arranging and linking icons on the CRT. Each icon represents and executes a specific function for data collection, analysis—from simple mathematics to complex algorithms—and presentation. You don't have to write a single line of code.

There are two HP VEE software packages for prototyping, experimentation, and problem modeling. HP VEE-Engine, at $995*, is a general-purpose tool for analysis and presentation of existing data. HP VEE-Test includes HP VEE-Engine and adds extensive I/O capability, including soft panels and device I/O objects for $5,000*.

So, if programming is keeping you from finding solutions, call **1-800-752-0900.** Ask for **Ext. 2380**, and find out how HP VEE clears the way.

* U.S. list prices.

There is a better way.

HEWLETT PACKARD

CIRCLE NO. 4

EDN Magazine Edition

June 20, 1991

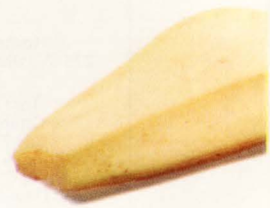
ELECTRONIC TECHNOLOGY FOR ENGINEERS AND ENGINEERING MANAGERS

SOFTWARE ENGINEERING SPECIAL SUPPLEMENT

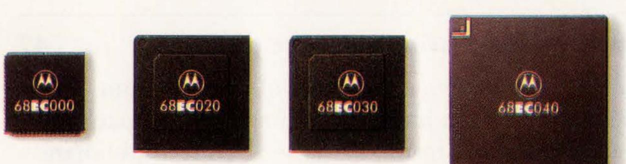
As software gains in importance, complexity, and cost, the need for information about software also increases. This special software engineering supplement addresses matters of concern to those who design and develop software for electronic products and systems. (Co er photo courtesy National Instruments)

SI ECIAL REPORT

DEPARTMENTS



OTHER 32-BIT EMBE
OFFER SOMETHI



When you're picking an embedded processor, it pays to watch your step. Or you may wind up with a design that doesn't meet your performance expectations, and a schedule that keeps slipping.

Not so with the new Motorola 68EC0x0 line.

From the 68EC000 that's just two dollars and change, to the 68EC040 that delivers a full 22 MIPS, they're all based on the world's most popular architecture. The 68000.

Which means you get a seamless migration path and reams of reusable code. Because each 'EC0x0 embedded microprocessor is binary compatible with all 68000-based microprocessors, and with every other 'EC0x0 family member.

What's more, they're all engineered to give

**Sample supplies are limited. Although the 'EC040 is not yet available, you can start your 'EC040-based design today simply by ordering the 68040. Motorola and the Ⓜ are registered trademarks*

DDED PROCESSORS NG OURS DON'T.

your products virtual immunity from memory wait states.

They also deliver superior levels of sustained performance, not "peak" MIPS like with other processors. So you can use DRAM instead of SRAM, and minimize your overall system cost.

68000 MICROPROCESSOR FAMILIES					
680x0 CPUs Architectural Integration	000	020	030	040	
68EC0x0 EMBEDDED Performance/Cost	EC000	EC020	EC030	EC040	
683xx INTEGRATED Functional Integration	302	330	331	332	340

Motorola's 68000 families let you choose the performance and integration that's right for your application.

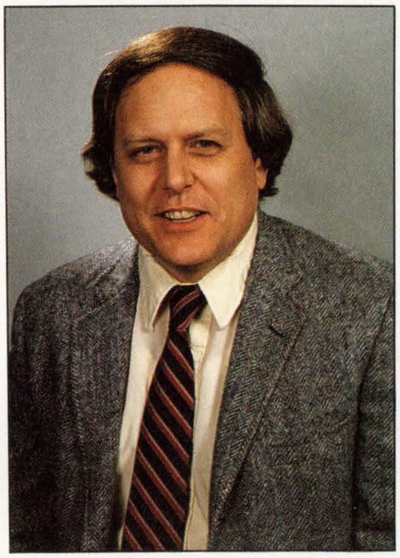
As you'd expect from Motorola, high performance, low cost and exceptional technical support are all part of the package. And 'EC0x0 embedded microprocessors are available at the price/performance points you need.

So before you pick your next embedded microprocessor, call 1-800-845-MOTO, and get a free sample from the 68EC0x0 family.* It's one step you'll never regret.

MOTOROLA

EDITORIAL

Intelligences theory reshapes thought



Textual programming—either you love it or your hate it. Diagrammatic programming—either you love it or you hate it. A new theory of multiple intelligences illuminates why some people feel instantly at home with diagrammatic programming and some do not. As noted researcher Howard Gardner points out, "intelligence" has a specific meaning in Western culture. To Westerners, an "intelligent" person is quick witted, wise, or rational. Being good at working with diagrams does not figure into the conventional Western notion of intelligence.

Looking for a quick way to categorize people, the educational establishment seized upon IQ tests. Misuse of IQ tests fostered the erroneous impression that only a single kind of intelligence exists and that a single metric can quantify that intelligence. Gardner's framework immediately exposes the weakness of conventional IQ testing. Such tests actually measure little more than mathematical and language ability.

Gardner postulates multiple intelligences. To find evidence of a separable intelligence within the human mind he looks for extraordinary abilities of individuals who are otherwise severely retarded and the abilities of precocious children. Mathematical, musical, and language abilities all fill the bill as separable intelligences. Both idiot savants and precocious children with these abilities are well known. Less well known, but just as easy to demonstrate as mathematical, musical, and language ability, is spatial intelligence. And spatial intelligence is the key to the power of diagrammatic programming.

Everyone knows that engineers typically rank in the upper few percentiles in mathematical ability. Less well known is that engineers rank in those same lofty percentiles in spatial intelligence. Alas, spatial and literary ability seldom go hand in hand. Engineers rank in the thirtieth percentile or lower in language ability. In fact, engineers are notorious for their inability to express themselves in words.

Rather, many engineers think and express themselves visually. Engineers' tools and methods require no words. Engineers' extraordinary spatial intelligence allows them to create designs and solve problems in ways that no computer can duplicate and, unfortunately, in ways that ordinary people cannot appreciate. No words can express how the power, beauty, and elegance of a good design stirs the passion of an engineer.

Because children having extraordinary spatial intelligence often cannot spell well or forget to dot their i's, the educational establishment labels them as dyslexic. Education becomes a barrier such children must surmount to reach an engineering career. Who knows how many potentially excellent, but unliterary, engineers the educational establishment glibly weeded out?

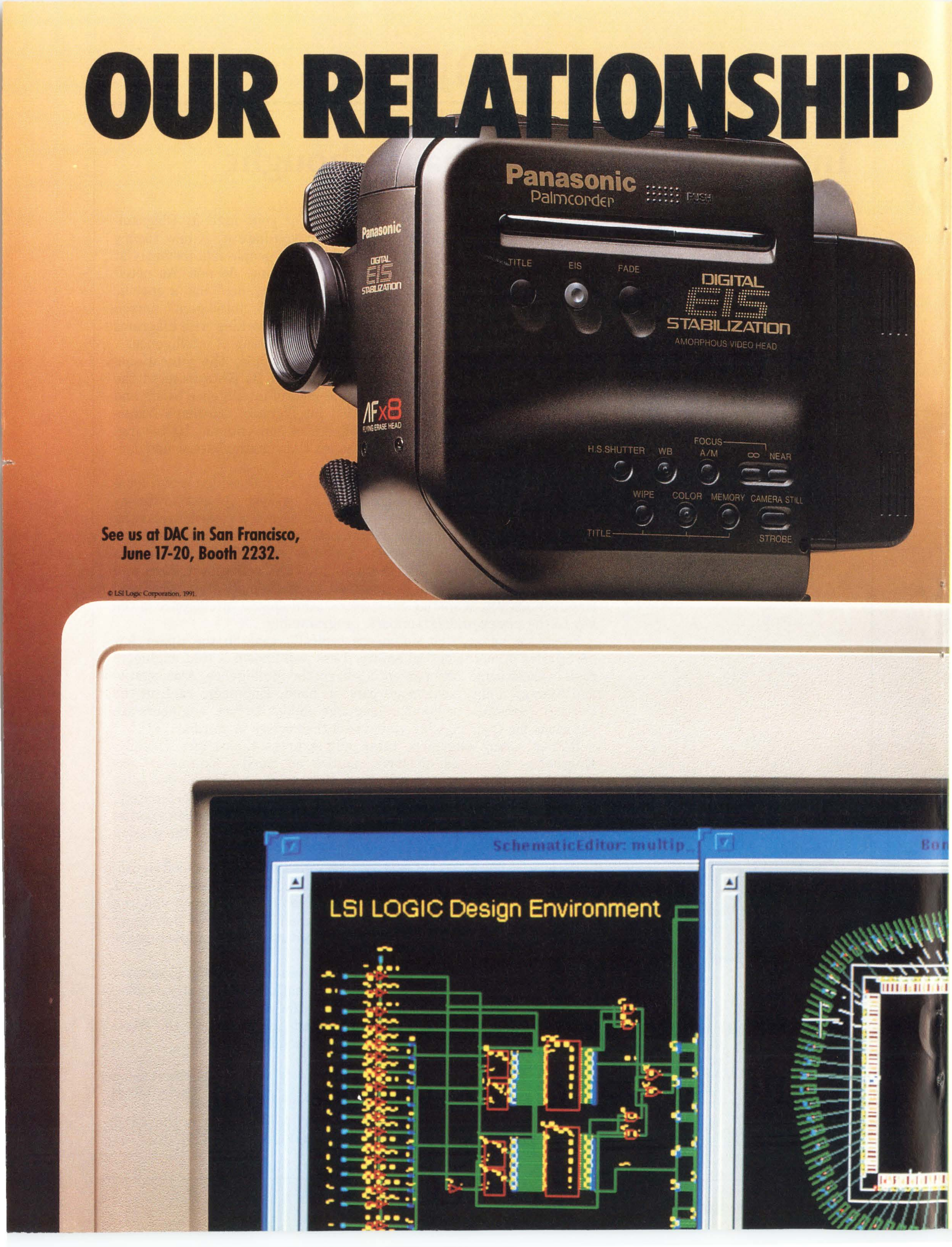
Gardner's Project Zero at the Harvard Graduate School of Education (Cambridge, MA) wants to change this situation. Among other tasks, Project Zero is developing tests for all of the seven intelligences that Gardner postulates. If Project Zero succeeds, it will change the Western view of intelligence and the way we teach our children.

Charles H Small
Senior Editor

Send me your comments via FAX at (617) 558-4470, or on the EDN Bulletin Board System at (617) 558-4241 300/1200/2400, 8, N, 1.

Jesse H. Neal
Editorial Achievement Awards
1990 Certificate, Best Editorial
1990 Certificate, Best Series
1987, 1981 (2), 1978 (2),
1977, 1976, 1975

American Society of
Business Press Editors Award
1988, 1983, 1981


OUR RELATIONSHIP
Panasonic
Palmcorder
TITLE
EIS
FADE
DIGITAL
EIS
STABILIZATION
AMORPHOUS VIDEO HEAD
Panasonic
DIGITAL
EIS
STABILIZATION
AFx8
FLYING ERASE HEAD
H.S.SHUTTER
WB
FOCUS
A/M
∞
NEAR
WIPE
COLOR
MEMORY
CAMERA STILL
TITLE
STROBE
See us at DAC in San Francisco,
June 17-20, Booth 2232.
© LSI Logic Corporation, 1991.
SchematicEditor: multip...
LSI LOGIC Design Environment

GOT VERY FUZZY. VERY FAST.

TOGETHER, MATSUSHITA AND LSI LOGIC GAVE THE NEW PANASONIC PALMCORDER A CLEAR ADVANTAGE: FUZZY LOGIC.

The market: volatile and changing fast. The products: getting smaller. And doing more. The competition: tough. The potential: significant worldwide sales gains from volume production of a superior camcorder.

No wonder Matsushita designers chose to work with LSI Logic to help create the cell-based ASIC chips for the new Panasonic Palmcorder.™

Our unique expertise in ASIC design tools and technology not only helped Matsushita make the new Palmcorders dramatically smaller, but helped add remarkable new capabilities as well.

Including a new image stabilization system based on fuzzy logic.

And everything was done in record time. From start of design to volume worldwide production, Matsushita and LSI Logic created each of the two key ASIC Palmcorder chips in less than 5 months.

We can do the same for you. LSI Logic offers the design tools, engineering expertise, and worldwide manufacturing capability to help bring your new and improved electronic products to market on time.

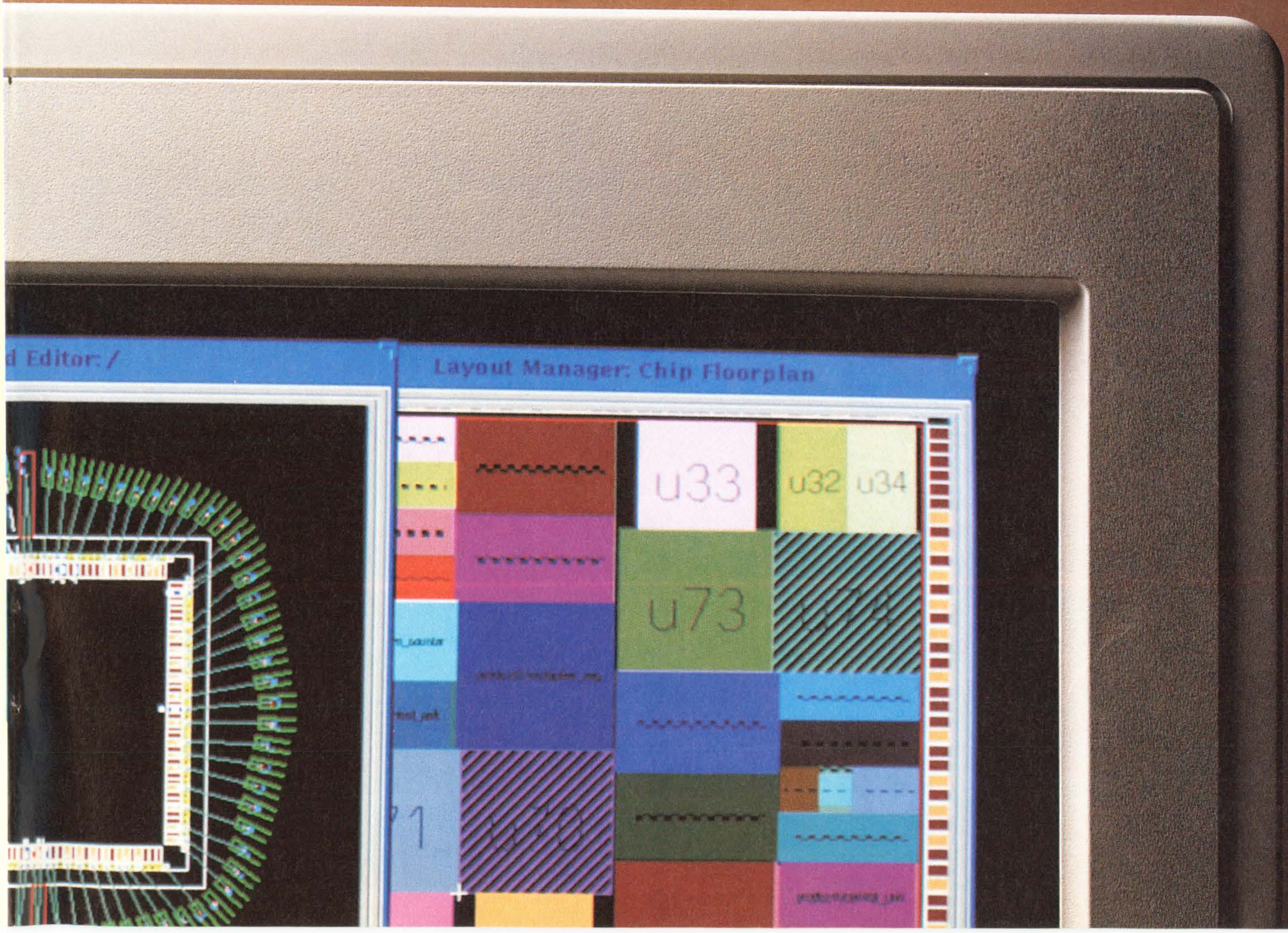
And in volume.

Call us at 1-800-451-2742 or write to LSI Logic, 1551 McCarthy Blvd., MS D102, Milpitas, CA 95035.

LSI LOGIC®

ACROSS THE BOARD

CIRCLE NO. 16



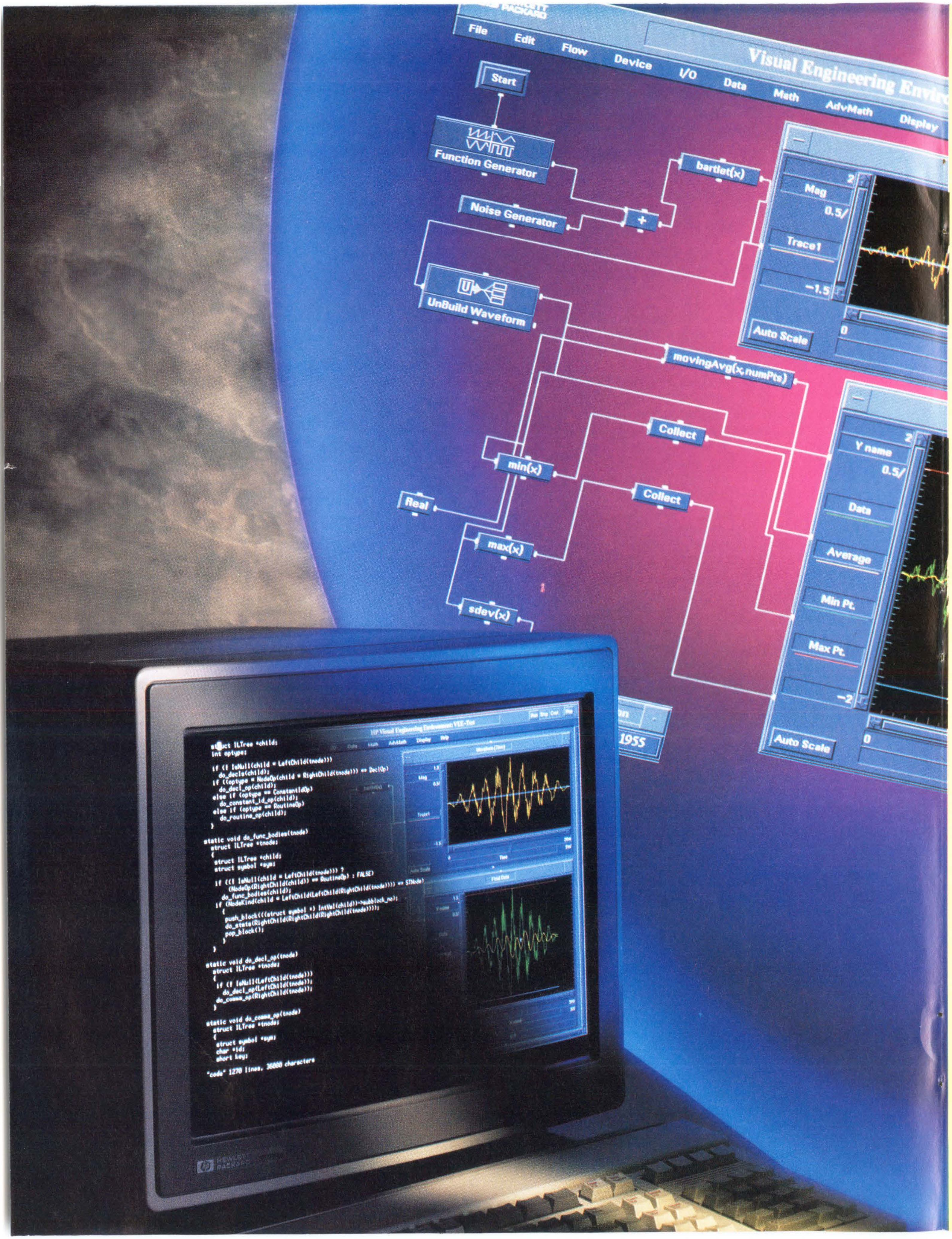
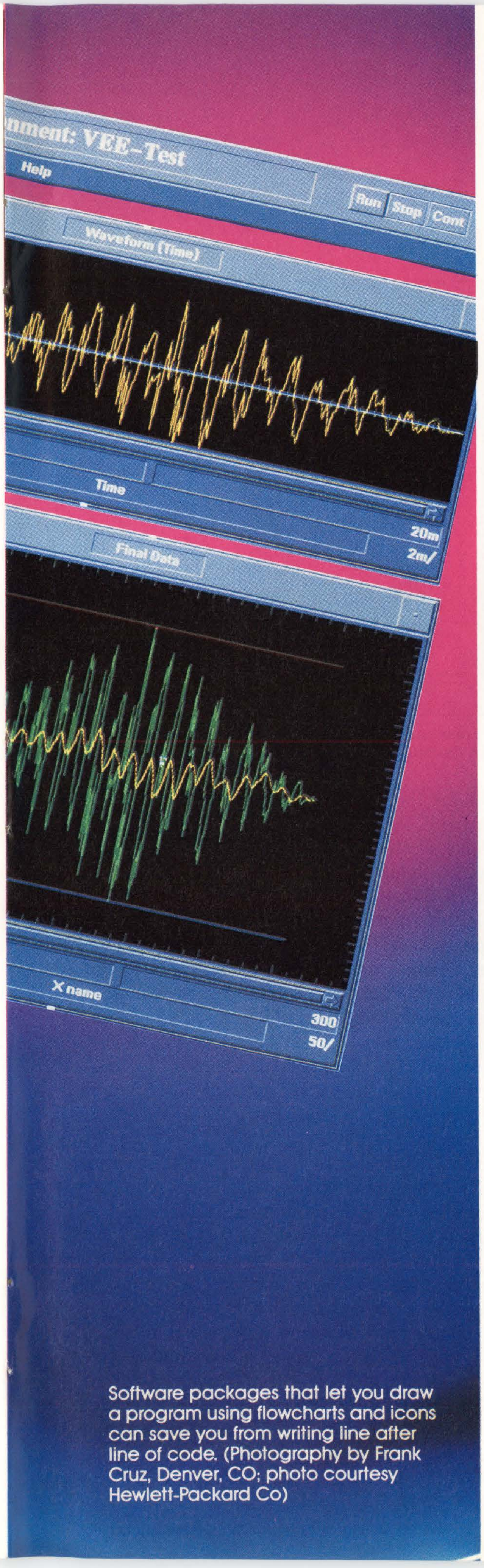

File
Edit
Flow
Device
I/O
Data
Math
AdvMath
Display
Start
Function Generator
Noise Generator
+
bartlet(x)
UnBuild Waveform
movingAvg(x,numPts)
Collect
Collect
min(x)
Real
max(x)
sdev(x)
Mag
Trace1
Auto Scale
Y name
Data
Average
Min Pt.
Max Pt.
Auto Scale

Diagram compilers turn pictures into programs



Software packages that let you draw a program using flowcharts and icons can save you from writing line after line of code. (Photography by Frank Cruz, Denver, CO; photo courtesy Hewlett-Packard Co)

On the fringes of computing, far from where the mainstream of computer science flows, pioneers are developing a radically different way to program computers. No longer does programming mean scrawling down endless linear lists of arcane, incomprehensible gibberish. Now you can draw your program.

Charles H Small, Senior Editor

By using diagrammatic programming to draw your program, rather than write it, you can create what the computer-science mainstream has vainly sought for years: comprehensible software. Diagrammatic programming produces comprehensible software because it taps the unsurpassed creative and analytical power of the visual hemisphere of the human brain. Simply stated, although humans cannot even begin to approach computers' text-handling expertise, they can still easily beat computers' pattern-recognition abilities. Drawing programs is so new, and its applications so diverse, that the technique has no formal name. You'll see the technique called, variously, pictorial programming, graphical programming, iconic programming, or diagrammatic programming.

The difference between text-based programming and diagrammatic programming is the difference between a description and a mug shot, between a list of directions and a map, between a formula and a graph. In diagrammatic programming, the ideal is the real; the documentation is the program.

Less than a trend, but far more substantial than a mere possibility, diagrammatic programming promises to do to textual programming methods what the internal-combustion engine did to the horse: Like horses, text-based programming will become either an expensive diversion for the rich or a last resort for the truly desperate.

Diagrammatic programming is inherently better than text-based programming for two reasons: Humans, especially engineers, can generate and comprehend pictures much more easily than they can linear lists of text. Secondly, the linear structure of a text-based program simply does not match the structure of

If programmers used diagrammatic programming, dead code would be as unlikely as civil engineers' drawing up blueprints for a road to nowhere.

real programs. A multidimensional picture can model a complex program much more elegantly and succinctly than any linear list can.

Still going down the wrong road

Periodically, the computer-science mainstream proclaims yet another variation on text-based programming that will magically transform programming from a craft into an engineering discipline. Remember "structured programming?" The latest such fad is "object-oriented" programming. But because these variations depend on human beings' reading, writing, and text-comprehension abilities, the latest, hottest textual programming method never seems to pan out. Progress in hardware is making the situation worse. Advances in hardware engineering more than nullify any advances in software engineering. **Fig 1** shows how hardware power has been increasing exponentially while software power has tended to level off.

A common and effective optimizing technique points out how incomprehensible large programs really are. All optimizing compilers perform "dead-code removal" as part of their bag of tricks. During dead-code removal, the compiler searches for and deletes all subroutines and functions that are never called. Compiler writers include this optimization technique because it is simple for a computer to perform and because experience shows that all large programs are riddled with dead code like raisins in an oatmeal cookie.

Think for a minute: How does dead code come about? Clearly, at one point in time, some programmer took the time to craft a function or subroutine. The programmer would not have written that code for the fun of it. The programmer must have felt the code to be necessary. Yet other programmers working on the project who could have taken advantage of that code either forgot about its existence or never needed it.

If programmers used diagrammatic programming instead of text-based programming, dead code would be as unlikely as civil engineers' drawing up blueprints for a road to nowhere.

A telling comparison

Comparing engineers' traditional graphical tools with programmer's development tools proves telling. Engineers' diagrams have two characteristics that set them apart from the doodles with which others design. Consider such graphical tools as Smith charts, Bode plots, pole-zero plots, and root-locus plots. Not only are these engineer's tools intensely graphical, but sup-

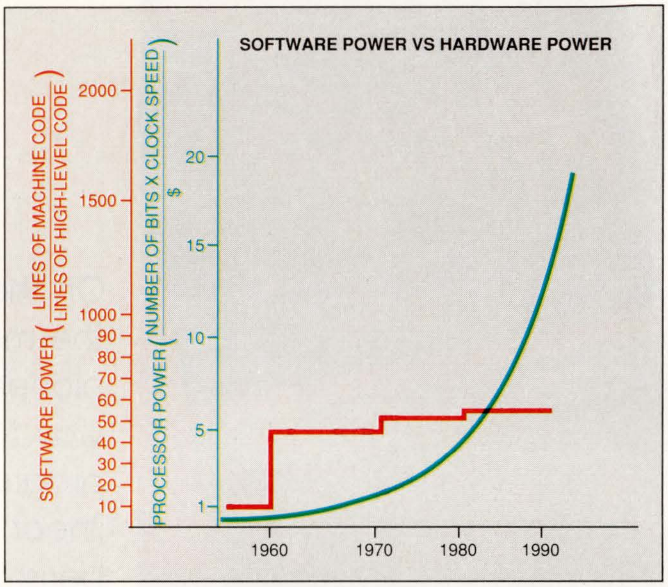


Fig 1—The crescendo of hardware improvements continues to overwhelm the hushed advancements in software practice.

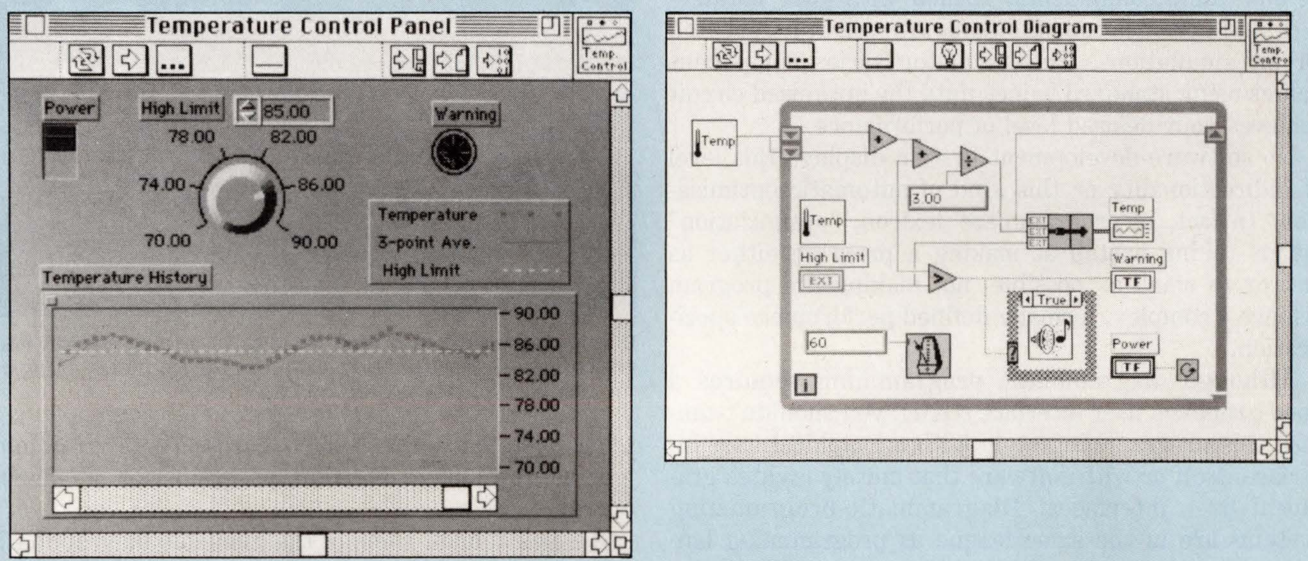
pressed beneath the graphics are powerful, formal systems of mathematics.

The second distinguishing characteristic of engineers' graphical tools is that they allow easy, bidirectional travel between the real and the ideal domains. Take a pole-zero plot, for example. An engineer can analyze a physical system and extract a pole-zero plot. At a glance, an engineer can tell much about the system from its pole-zero plot. And this process is reversible. An engineer can dispose poles and zeros on a pole-zero plot to create an ideal system that has the desired performance characteristics. From this plot, the engineer can synthesize the specs for real components, which will yield a system that performs according to the idealized pole-zero plot.

Conventional software tools do not exhibit these characteristics. The tools do not rest on powerful, formal mathematics and are not bidirectional.

For example, although utilities are available that will extract a flow chart from a program, you cannot modify such an extracted flow chart and recompile it. You can diagram a program, but—until recently—you still had to concoct linear lists of unfathomable text to realize that program. Now, using diagrammatic programming, you can actually compile your diagram. Once you can compile a software diagram, the need to go back and forth between the ideal and real domains largely goes away.

Jensen Transformers' Comtran ($950 to $3850) illustrates the virtue of basing your diagrammatic-programming system on formal, mathematically defined engineering tools. This software runs on Hewlett-Packard desktop computers and IBM PCs equipped with an HP Basic Language Processor card ($3000). It exhibits the bidirectional nature of engineers' graphical tools. Using Comtran, you can program IEEE-488 instruments to measure and analyze the performance of a real-world system. The software presents the results of its analyses as graphs in the ideal domain.



(a)

```
#include "WindowSupport.h"      /* headers supporting window and display objects */
#include "ChartSupport.h"
#include "SwitchSupport.h"
#include "LEDSupport.h"
#include "KnobSupport.h"
#include "TimeSupport.h"
#include "DataAcqSupport.h"     /* header for data acquisition board support */

int boardNum = 1;
int channelNum = 1;

int ReadTemp(void);             /* prototype for function that reads temperature */

main()
{
    windowPtr          windowP;
    multiplotChartPtr  multiplotChartP;
    switchPtr          switchP;
    LEDPtr             ledP;
    KnobPtr            knobP;

    int         time;
    float       temp[3], avgTemp;
    float       limitValue;
    int         switchValue;

    int         numPlots = 3;
    float       plotPts[numPlots];

    windowP = NewWindow(...);                   /* create window and display objects */

    multiplotChart  = NewMultiplotChart(windowP, numPlots,...);
    switchP         = NewSwitch(windowP, ...);
    ledP            = NewLED(windowP, ...);
    knobP           = NewKnob(windowP, ...);

    SetSwitch(switchP, True);                   /* initially, the switch is up */

    temp[0] = ReadTemp();
    temp[1] = temp[0];

    do {                                        /* loop, reading the temp once a second */
        time = CurrentTimeInTicks();            /* and displaying the value until */
                                                /* the switch is turned off  */
        temp[2] = ReadTemp();
        limitValue = ReadKnob();

        avgTemp = (temp[0]+temp[1]+temp[2])/3.0;

        plotPts[0] = temp[2];
        plotPts[1] = avgTemp;
        plotPts[2] = limitValue;

        AddPtsToMultiplotChart(multiplotChartP, plotPts);

        if (temp[2]>limitValue) {
            SetLED(ledP, True);
            Beep();
        }
        else
            SetLED(ledP, False);

        temp[0] = temp[1];
        temp[1] = temp[2];

        while (CurrentTimeInTicks() != time + 60)
            ; /* Pause before new reading */

        switchValue = ReadSwitch();
    } while (switchValue == True);

    DisposeKnob(knobP);                         /* free up memory used by window */
    DisposeLED(ledp);                           /* and display objects */
    DisposeSwitch(switchP);
    DisposeMultiplotChart(multiplotChartP);
    DisposeWindow(windowP);
}

float ReadTemp(void) {                          /* read a value from the board */
    int      reading;                           /* and convert it to a voltage */
    float    voltage;

    AIRead(boardNum,channelNum, gain, &reading);
    AIScale(boardNum, gain, reading, &voltage);
    return voltage;
}
```

(b)

Using diagrammatic programming, you can easily create and operate programs. A diagrammatic program for measuring, analyzing, and displaying temperature along with a simulated panel for controlling the program is shown in **a**. The wide arrow surrounding the diagrammatic program indicates a loop. A text-based C program for the same task is shown in **b**. (Photo courtesy National Instruments)

A multidimensional picture can model a complex program much more elegantly and succinctly than any linear list can.

These graphs have the proven and powerful formats familiar to all analog-design engineers.

To progress from the ideal to the real domain, Comtran also performs analog simulation. You program the simulator by drawing circuit diagrams. Comtran's circuit "icons" correspond to mathematically defined ideal components. Armed with your parameters and specification, the software will run repeated circuit simulations. The simulator varies component values using standard values until the optimized circuit achieves your desired level of performance.

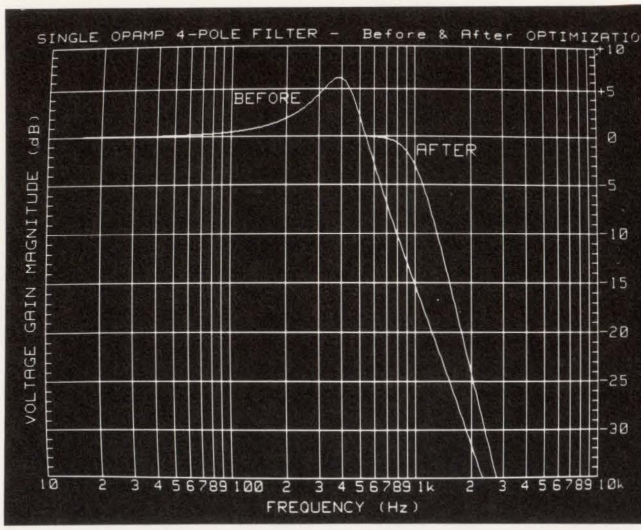
No software-development system displays this level of bidirectionality or this kind of automatic optimization. In fact, in programmers' lexicon, "optimization" means taking a stab at making a program either as fast or as small as possible, not making the program achieve a complex, formally defined performance specification.

Although diagrammatic programming requires a good graphical user interface (GUI), you shouldn't confuse this programming style with a graphical user interface itself or with software that merely creates graphical user interfaces. Diagrammatic-programming systems are in the same league as programming languages.

Obviously, if you are going to draw your program, the computer you are using must have high-resolution graphics. Indeed, the inspiration for many diagrammatic-programming systems was the Apple Macintosh's high-resolution graphics and graphics-oriented operating system. Without a host computer such as the Mac or a Unix computer running MIT's (Cambridge, MA) X-Window System, crafting a diagrammatic-programming system would be difficult. Because of the IBM PC's clunky graphics, diagrammatic-programming systems for these computers were the last to emerge.

Diagrammatic programming can make what were once difficult software tasks easy. Real-time software is—with the possible exception of digital-signal-processing software—probably the most difficult kind of software to write, debug, and verify. Diagrammatic programming makes developing multitasking or multithreaded programs effortless. You simply draw two lines of execution.

The diagrammatic-programming pioneers are not on an altruistic mission to save the world from bad programs. Their immediate goals are much more mundane. Many are trying to sell program-development or simulation systems to people who don't like programming very much. Several diagrammatic-programming systems address such hard-core programming jobs as developing real-time programs for single-chip microprocessors.



Displaying results in a form familiar to analog-design engineers, Jensen Transformers's Comtran automatically optimizes the response of a simulated analog circuit.

A quick tour of some of the available diagrammatic-programming systems discloses that they fall into four broad categories—with some products overlapping categories:

- Data-acquisition and instrument-control, and data-analysis and -display software
- Engineering simulators
- Special-purpose program generators
- General-purpose program generators.

National Instruments began work on its diagrammatic-programming system, Labview ($1995), in 1984, which qualifies the firm as a pioneer in diagrammatic programming.

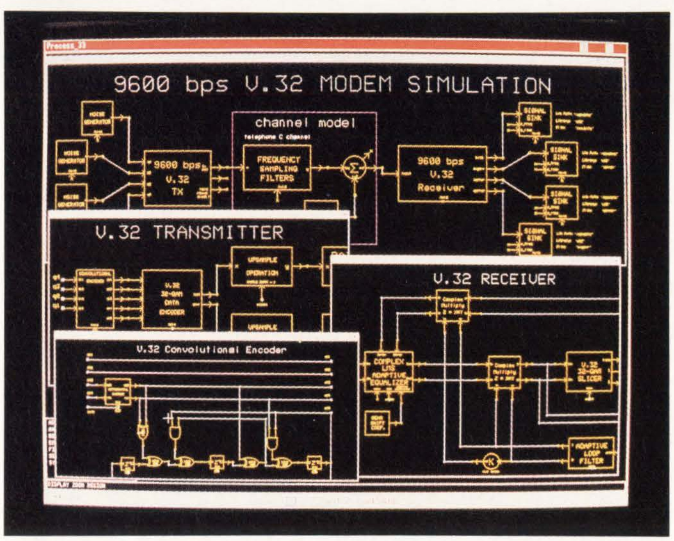
National Instruments chose the data-flow diagram as a visual metaphor. As with all diagrammatic-programming systems that animate a standard software-design method, the company's programmers had to extend their chosen software metaphor to make it a workable tool. As soon as the first prototype was working, the need for icons that performed looping and iterating became painfully apparent. These icons perform the functions of common software constructs such as WHILE, FOR, and DO loops. The firm also concocted control icons that correspond to CASE statements as well as an icon that impresses a sequential execution order on segments of the data-flow diagram.

The academic data-flow diagram, as presented in computer-science texts, omits these necessary operations. The developers at National Instruments, as well as all developers of diagrammatic-programming systems, had to add features to academic models.

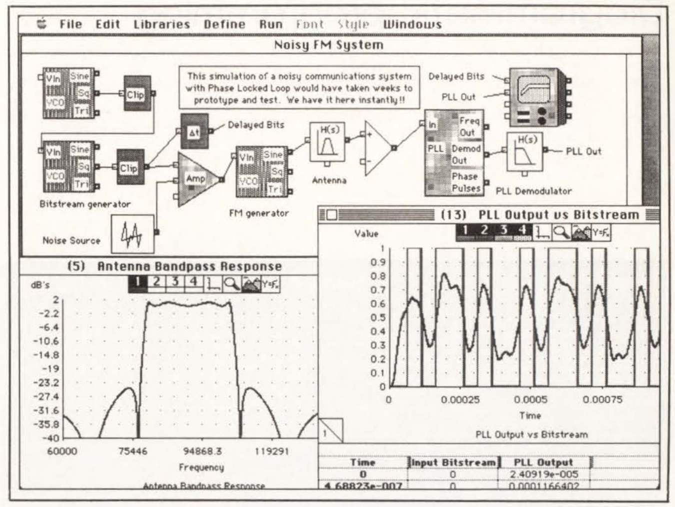
National Instruments developed Labview for engineers working on IEEE-488 test systems. However, the program accepts data from any source (not just IEEE-488 instruments), so you can use the software as either a general-purpose engineering-simulation or data-analysis tool.

Before diagrammatic-programming systems for IEEE-488 instruments, engineers programmed IEEE-488 hardware using versions of Basic. In the best of circumstances, Basic is an unwieldy tool for anything but the smallest and simplest mathematics-oriented tasks. Even with enhancements for handling real-time interrupts, writing an interrupt-driven, real-time instrument-control and data-analysis program in Basic is a mind-warping, nerve-shredding chore.

Controlling precision test instruments remotely over the IEEE-488 bus requires reams of idiosyncratic, low-level commands. Diagrammatic-programming systems for IEEE-488 instruments submerge this idiosyncratic boilerplate under simulated "front panels" the software displays on the host computer's screen. Mousing the controls of the simulated front panel causes the computer to output the appropriate command string to the instrument and, in the case of measuring instruments, to collect the results of the instrument's measurement. Diagrammatic-programming systems for IEEE-488 in-



As is common for diagrammatic-programming systems, Comdisco's Signal Processing Workstation hides nitty-gritty details in low-level, hierarchical windows. This screen shows three levels of a V.32bis modem: a block diagram of the entire modem, the receiver and transmitter sections, and—at the lowest level—details of the encoder block.



The Apple Macintosh personal computer spawned many diagrammatic-programming systems.

struments have built-in routines for analyzing and displaying data on simulated instrument readouts or in graphical form.

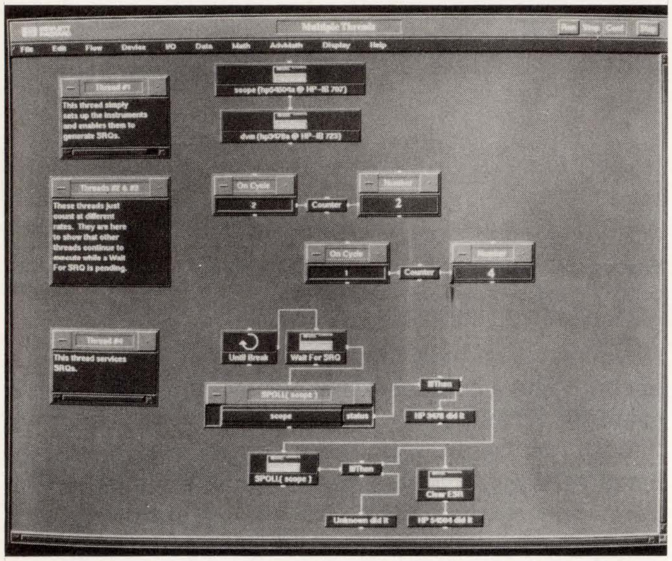
With Labview, as with other diagrammatic-programming systems, multitasking is effortless and crystal clear. If your application demands multitasking, you simply draw multiple streams of execution. Using text-based programming for real time results in murky code that rivals old-fashioned Fortran spaghetti code for incomprehensibility.

Wavetek's Wavetest-XTM ($7995) has evolved into a diagrammatic-programming system for IEEE-488 that is similar to Labview. This software runs on DEC workstations under DEC's VMS real-time operating system and uses the X-Window System for display. The software combines two visual metaphors: the data-acquisition portion of the software employs the flow chart as a visual metaphor, and the analysis portion—a product DEC developed—uses the data-flow diagram.

Wavetek had to add enhanced case, looping, and branching icons to the standard flow-chart symbols. The company also added a real-time construct to handle asynchronous service requests (SRQs) from IEEE-488 instruments. To set up a prioritized interrupt handler, you expand the SRQ icon and fill in a table. The table has spaces for identifying the interrupt sources, assigning a handler for their interrupts, and assigning a priority to each interrupt source. This simple table provides a clean and concise solution to what is a nasty bit of coding in other programming systems.

Also bridging data acquisition and display, as well as general-purpose simulation, is Hewlett-Packard's VEE (simulator $995; IEEE-488 version, $5000). This software runs on HP 9000 computers under HP-UX (HP's Unix) and the X-Window System. The visual metaphor for VEE is the data-flow diagram. You draw your program by connecting icons. The software comes with icons for engineering and mathematical operations.

The icons themselves are examples of object-oriented programming. For example, the simple-seeming multi-



Diagrammatic-programming systems make multitasking effortless. To launch two independent processes using Hewlett-Packard's VEE system, for example, you draw two separate threads of execution.

plication icon can perform any mathematically allowable multiplication. The icon can sense the types of data that appear at its front end. The icon can multiply integers, floating-point numbers, complex numbers, and matrices without specific instructions.

Extend ($495) from Imagine That Inc lets you simulate complex electronic systems on your Mac. This general-purpose, dynamic-system simulator comes with function-block icons for amplifiers, filters, digital gates, and control functions. The program also has data-analysis and display icons, so you can see the results of your simulation. Jasco Systems Ltd's Spawn software ($995) offers analogous possibilities for IBM PCs.

I-Logix Inc chose an enhanced state-transition diagram as its visual metaphor for diagrammatic-programming systems. The company's booklet describing the enhancements is worth reading if for no other reason than to improve your own manually drawn state-transition diagrams (**Ref 1**).

The company's first product, Statemate ($15,000 to $70,000), simulates finite-state machines. You draw a state-transition diagram and other supporting documents on the screen of your workstation. You can then link your state machine to a simulated operator's control panel and "operate" your state machine.

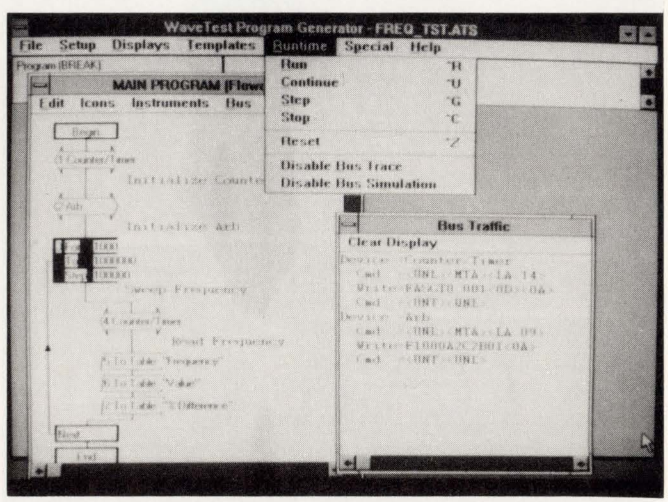
Following the trend of metamorphosing simulators into code generators, the company's latest product, Express VHDL ($32,500), turns your state machine into a hardware-description-language listing in VHDL (VHSIC Hardware Description Language). You can compile the resulting VHDL listing into an ASIC. In other words, you can go from an elegant and rarefied finite-state machine to an IC at the press of a function key.

Not all diagrammatic-programming systems are general-purpose simulators. Some address specific tasks such as digital-signal processing (DSP). Writing DSP software is a software engineer's nightmare. DSP combines arcane mathematical algorithms, abstruse physics, and quirky μP instructions sets into a horrific witch's brew.

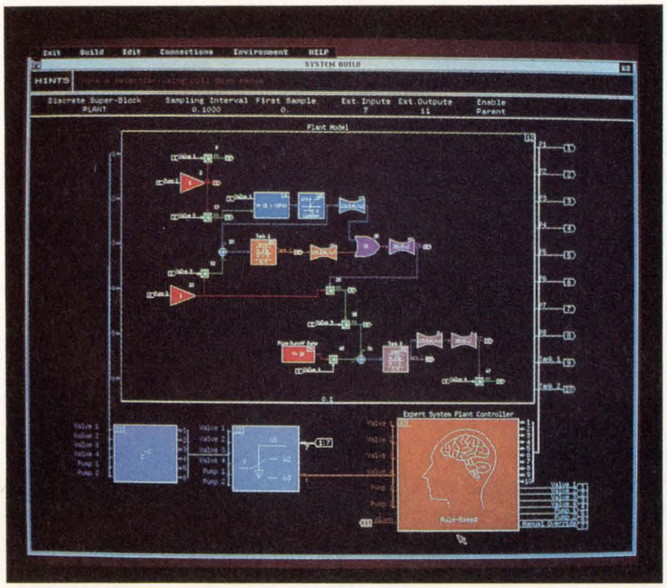
Comdisco Systems' Signal Processing Worksystem ($25,000), which runs on Unix workstations, began life as a simulator. The simulator accepts simulation programs drawn on a workstation screen using engineers' DSP function blocks. Comdisco's visual metaphor is the familiar engineers' block diagram. Now that C compilers are available for DSP μPs, the system has evolved into a program generator as well.

The essential task of DSP is molding waveforms into a desired form. Comdisco's tool can display and manipulate multiple waveforms, so you can compare your DSP system's inputs and outputs.

The software comes with libraries comprising hundreds of function blocks. The function blocks generate signals, filter signals, modulate and demodulate signals, encode and decode signals, and perform various mathematical and nonlinear operations. You wire these function blocks up in a diagram that superficially re-



All diagrammatic-programming systems based on existing software-design tools needed additional operations to become workable programming systems. When Wavetek chose the flow chart as the visual metaphor for its Wavetest-XTM program, the company had to add numerous icons for looping, selecting, and broadcasting.



A rule-based, real-time expert system is not beyond the scope of diagrammatic programming, as this I-Logix Statemate program shows.

sembles a data-flow diagram. However, DSP systems are clocked, pipelined systems, so the analogy is inexact. Data flow through a DSP diagram in lock step, each block performing its function at each clock pulse. In contrast, the blocks in a data-flow diagram, much like the synapses in your brain, do not fire unless a complete set of data arrives at a block's input.

Hyperception's Hypersignal ($489 to $2995) addresses digital-signal processing on IBM PCs in a similar fashion. The company also employs the block diagram as a visual metaphor. The software runs under DOS or Windows 3.0. This software is mainly a simulator; only some of the digital-filter blocks generate executable code in C and DSP-μP assembly language.

Integrated Systems' AC-100 ($25,000 to $120,000) is a diagrammatic-programming system designed for control engineers. The software runs on DEC VAXstations and accepts program specifications in the form of control engineers' block diagrams. The program creates a nonreal-time simulation from the block diagram. You can run this simulation from a simulated "front panel" display, observing the simulation's response. The program displays outputs in graphical forms familiar to control engineers, such as Bode plots, root-locus plots, and Nichols plots.

The software can also generate real-time programs, in C or Ada, for the firm's hardware simulator. This hardware simulator is a Multibus II backplane with various processor and I/O boards. The software can generate multiprocessor programs running on as many as ten 80386 CPU boards in the hardware simulator.

Even some of the proponents of object-oriented programming realize that diagrammatic programming is the best way to make programs comprehensible. However, diagrammatic programming alone does not make object-oriented programming clear. On the road to understanding object-oriented programming, the novice must first hurdle the considerable barrier that object-oriented programming's peculiar jargon forms.

Objectcraft's eponymously named Objectcraft program ($399) lets you construct a C++ program from icons that symbolize the various entities that populate object-oriented land. You can draw a sort of data-flow diagram having interconnected "objects" (programs or utilities). These objects can have "slots" (variables, arrays, or structures) for numbers, strings, and pointers. The objects can also contain "methods" (subroutines or functions).

You can "import" (link in) existing programs and utilities. Objectcraft (the program, not the company) then generates real C++ or Turbo Pascal 5.5 code from your diagram.

Semaphore Tools' Pilot software ($5000) lets you draw a diagram to structure a C++ program. Conversely, you can alter your C++ listing, and the tool will extract an updated software diagram from your altered diagram.

If you have a hankering to try diagrammatic programming, you are not confined to grandiose, high-level projects such as designing a VHDL ASIC or simulating a communications network. Tao Research Corp's GDS-11 ($1725) is a diagrammatic-programming system for the 68HC11 single-chip microprocessor. The software includes a compiler, simulator, and symbolic debugger and runs on the Macintosh. This system uses state-transition diagrams as a visual metaphor. The state diagrams follow usual engineering practice with a few extensions. One minor change is that the states are in rectangular boxes instead of ovals. Boxes are, after all, easier for the computer to draw than ovals.

A more important change is that you can subsume a collection of states and their relationships under a single state. That is, you can hierarchically rank submachines beneath a parent state machine. This facility proves handy when a state machine becomes too big or unwieldy to display on the Mac's screen.

The software's state diagrams conflate elements of Mealy and Moore finite-state machines. Hardware engineers have no problem distinguishing between Mealy and Moore finite-state machines, but programmers usually have a dimmer understanding of formal finite-state automata. In keeping with Mealy state machines, you annotate each directed state transition with the input that triggers the transition and—separated by a slash—the output that accompanies that transition. But like a Moore state machine, you can also put an action in the state box. The program will execute this action as long as the program is in that state.

Just like other diagrammatic-programming systems, multitasking is effortless with GDS-11. You draw a state-transition diagram for each independent process. The diagrammatic-programming compiler produces both intermediate code for the system's simulator and 68HC11 machine code in Motorola S records.

In the short term, expect more established simulation and analysis tools to acquire diagrammatic-programming front ends. The Mathworks's Simulab, Tutsim Products' (Palo Alto, CA) Tutsim, and several enhanced versions to Spice have already gained such user-friendly front ends. In the long term, diagrammatic programming could be a paradigm shift for programming.

Writing, debugging, and verifying a text-based program is the devil's own curse. Few humans have the ability and the inclination to engage in such a hassle. With luck, these more-humane pictorial programming tools will become prevalent, leaving text-based programming to those chosen few who have the kinds of minds that can handle such a chore. After all, we'll always need a few text-based programmers to write compilers. Further, diagrammatic programming offers the exciting possibility of automating the construction and verification of programs, much as pc-board-layout and ASIC tools perform automated place-and-route and minimization. EDN

References

1. *The Languages of Statemate,* STCON-01, i-Logix Co, Burlington, MA, November 1987.
2. Gardner, Howard, "Frames of Mind, The Theory of Multiple Intelligences," Basic Books, New York, NY, 1985.
3. Gardner, Howard, "The Mind's New Science: A History of Cognitive Revolution," Basic Books, New York, NY, 1987.
4. Gardner, Howard, Letter to author, March 25, 1991.

Article Interest Quotient (Circle One)
High 488 Medium 489 Low 490

Manufacturers of diagrammatic-programming systems

For more information on diagrammatic-programming systems such as those described in this article, circle the appropriate numbers on the Information Retrieval Service card or use EDN's Express Request service. When you contact any of the following manufacturers directly, please let them know you saw their products in EDN.

Comdisco Systems Inc
919 E Hillsdale Blvd
Foster City, CA 94404
(415) 574-5800
FAX (415) 358-3601
Circle No. 650

Hewlett-Packard Co
Box 301
Loveland, CO 80539
(303) 679-5000
Circle No. 651

Hyperception Inc
9550 Skillman LB 125
Dallas, TX 75243
(214) 343-8525
FAX (214) 343-2457
Circle No. 652

I-Logix Inc
22 Third Ave
Burlington, MA 01803
(617) 272-8090
FAX (617) 272-8035
Circle No. 653

Imagine That Inc
151 Bernal Rd, Suite 5
San Jose, CA 95119
(408) 365-0305
FAX (408) 629-1251
Circle No. 654

Integrated Systems Inc
2500 Mission College Blvd
Santa Clara, CA 95054
(408) 980-1500
Circle No. 655

Jasco Systems Ltd
9865 W Saanich Rd
Sidney, BC
Canada V8L 3S1
(604) 656-0234
FAX (604) 656-5833
Circle No. 656

Jensen Transformers Inc
10735 Burbank Blvd
North Hollywood, CA 91601
(213) 876-0059
FAX (818) 763-4574
Circle No. 657

National Instruments Co
6504 Bridge Point Pkwy
Austin, TX 78730
(512) 794-0110
FAX (512) 794-8411
Circle No. 658

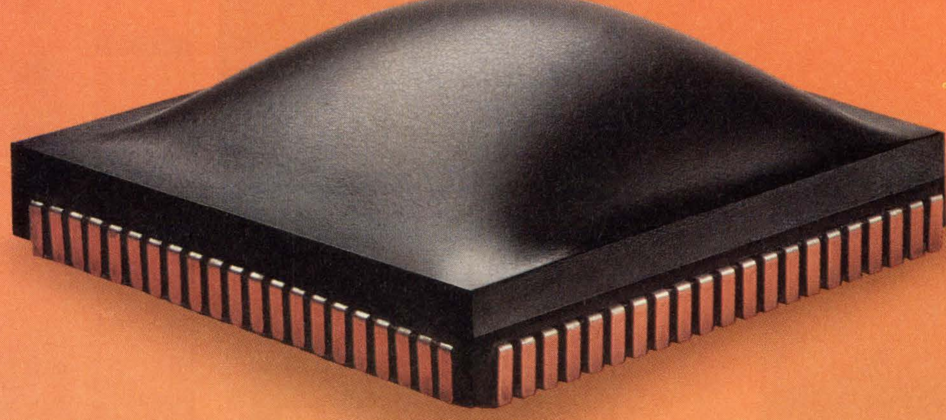
Objectcraft
2124 Kittredge St, Suite 118
Berkeley, CA 94704
(415) 540-4889
FAX (415) 861-5398
Circle No. 659

Semaphore Tools
800 Turnpike St, Suite 200
North Andover, MA 01845
(508) 686-9850
Circle No. 660

Tao Research Corp
39182 Mission Blvd, Suite 205
Fremont, CA 94539
(800) 324-6020;
in CA, (415) 770-1344
Circle No. 661

The MathWorks
24 Prime Park Way
South Natick, MA 01760
(508) 653-1415
Circle No. 662

Wavetek Corp
Box 85434
San Diego, CA 92138
(619) 450-9971
Circle No. 663



Pack more logic into every FPGA.

NEW ABEL-FPGA helps you get the most out of the latest FPGAs. If you want to take advantage of the sophisticated capabilities of today's FPGAs, only Data I/O®'s new ABEL-FPGA™ Design Software has the power to pack in maximum logic. It combines the industry-standard ABEL Hardware Description Language (ABEL-HDL™) with our new intelligent FPGA Device Fitter™ technology. So, you can create more complex designs with less effort—ABEL-FPGA does the hard work for you!

ABEL-FPGA's powerful Device Fitters automatically optimize your circuits for minimum area or maximum speed. Fitters are available for all the leading architectures, including Actel, Altera, AMD, Atmel, ICT, National, Plus Logic, and Xilinx. And with built-in knowledge of its target architecture, each fitter masters the complex features of its device automatically, intelligently.

Practical, detailed documentation, complete with FPGA design examples, also helps to ensure that you get the most from each architecture. And for added design power and flexibility, ABEL-FPGA lets you specify place-and-route constraints directly in your circuit description, so you can easily migrate the same design between multiple FPGA vendors.

Pack more logic into your next FPGA design, with the single solution to all your FPGA behavioral entry needs: ABEL-FPGA. **Call today to find out more about NEW ABEL-FPGA.**

1-800-247-5700

NEW ABEL-FPGA with FPGA Device Fitters!

The Personal Silicon Experts

DATA I/O Corporation

CIRCLE NO. 9

ROM monitor tips and tricks aid debugging effort

In-circuit emulators have supplanted the primitive ROM monitors of years past, but new high-speed microprocessors are often a step ahead of emulators' capabilities. So, once again, ROM monitors—basic and inexpensive, but no longer primitive—are gaining favor. Though not without limitations, ROM monitors can do a lot if you know how to use them.

Peter Dawson, *Embedded Support Tools Corp* and Andy Lantz, *Intermetrics Microsystems Software Inc*

Engineers debugging software for the latest, fastest, and most complex microprocessors lack the necessary tools. In-circuit emulators for the chips aren't always available as soon as the silicon is; and even if emulators are available, they don't always do everything you want them to. An emulator might not run at a processor's full speed, for example, because its intrusive nature can alter a computer board's electrical characteristics and make full-speed emulation unreliable.

New versions of an old debugging tool—the ROM monitor—can help you deal with these problems. A ROM monitor is simply an EPROM-resident program on a computer board that gives you certain debugging abilities (see **box**, "ROM monitors: Basic but improved," pg 24). ROM monitors can't do everything that in-circuit emulators can, but their limitations aren't as great as you might think. You can overcome many of the limitations by using the right approach and by combining ROM monitors with logic analyzers and ROM emulators.

Misperceptions about limitations

Among the limitations that some software developers perceive about ROM monitors are:

- Application code won't run in real time, because the application code and the ROM monitor's code must share the same processor.
- You can't acquire a trace of program activity because a ROM monitor—unlike an in-circuit emulator—doesn't observe bus activity.
- You can't set breakpoints on instructions in ROM because a ROM monitor doesn't observe bus activity and because code in ROM is unalterable.

These perceptions fail to consider some of a ROM monitor's abilities, however. ROM monitors can work around each of these situations and provide additional functions as well (see **box**, "How a ROM monitor works," pg 26).

The concern about application code running in real time does sound reasonable at first. After all, your application code and the ROM monitor's code do share the same processor. Between breakpoints, however, your code runs in real time; once a ROM monitor passes control to your application, the monitor is inactive until your code encounters the next breakpoint. And, if your target processor has on-chip cache memory, your code can utilize the cache at full speed. In-circuit emulators sometimes have to disable the cache in order to perform reliably.

In some cases, ROM monitors are faster than in-circuit emulators.

At times, though, you will want to sacrifice real-time speed in order to use additional debugging features. For example, your code slows down significantly (a factor of 8 is typical) if you instruct the ROM monitor to accumulate a record of program activity in a trace buffer. It is possible, though, to use a logic analyzer with a ROM monitor and not suffer this speed penalty.

If you use a ROM monitor alone, filling a trace buffer will be slower, but still possible. Although a ROM monitor has no hardware for monitoring bus activity to observe a microprocessor's program counter, it works around this limitation by taking advantage of the processor's trace mode.

A ROM monitor initiates the trace mode by setting a bit in the microprocessor's status register. In trace mode, a trace exception (a software interrupt) occurs after each instruction; the exception-handler code then invokes the ROM monitor, which stores the program counter of the instruction just executed in a RAM table. (The trace vector pointing to the exception handler is one resource that you must reserve for the ROM monitor's use. See **box**, "Some practical matters," pg 86 .)

ROM monitors: basic but improved

Before in-circuit emulators there were ROM monitors—simple ROM-resident programs that provided minimal debugging features. Then, as now, ROM monitors formed part of the memory space of a target system in need of debugging.

Early ROM monitors let you run your program with breakpoints on individual instructions or to step through your code one instruction at a time. You couldn't acquire an instruction-trace record and you couldn't set breakpoints on data references. You could display register contents, and you could display and alter the contents of memory locations, but user interfaces were at the machine-code level; displays were primitive, and you issued commands as numbers rather than mnemonics.

Better chips, better monitors

Today's ROM monitors are more sophisticated. Taking advantage of newer microprocessors' trace bits or trace flags, they provide instruction-by-instruction program tracing that you can enable and disable at points of your choosing. The processor's trace bits and flags also make possible data breakpoints—as opposed to code (instruction) breakpoints—so that your program will halt any time it changes a data location that you've specified. Your program does slow down significantly when you run your processor in trace mode, but between breakpoints with tracing disabled, your program will run in real time.

In some cases, ROM monitors are even faster than in-circuit emulators. The transmission-line effects of an emulator's cabling can alter a high-speed computer board's electrical characteristics, and thereby preclude the emulator's full-speed operation. Problems of this type are increasing as some microprocessors increase their pin counts and others move to fine-pitch surface-mount technology, thus making socketing more complicated. ROM monitors are unaffected by cabling, however; they reside on the board and make use of the board's actual processor.

Chips that operate with an internal-instruction cache or prefetch instructions also present problems for emulators that ROM monitors can circumvent. An emulator can only observe information on the bus—external to the chip—and because instructions in cache are internal to the chip, an emulator can't be aware of them. Therefore, if you use an emulator for a chip with an instruction cache, you must disable the cache to be able to debug precisely. A ROM monitor, however, can provide debugging with the cache enabled.

Special chips for ROM Monitors

As microprocessor manufacturers design more features into silicon, the use of ROM monitors will increase. Motorola's MC68332, for example, has a background mode that provides "hooks" into its operation. Intel's 80386 even has hardware breakpoint capabilities designed into the chip.

ROM monitors don't do everything that emulators do, but they are much cheaper—typically less than $1000, versus perhaps $15,000 for an emulator. Because of that extreme price difference, some software-development teams use several ROM monitors and one shared emulator. The ROM monitors work well for many phases of debugging; the emulator is reserved for tasks that really need it.

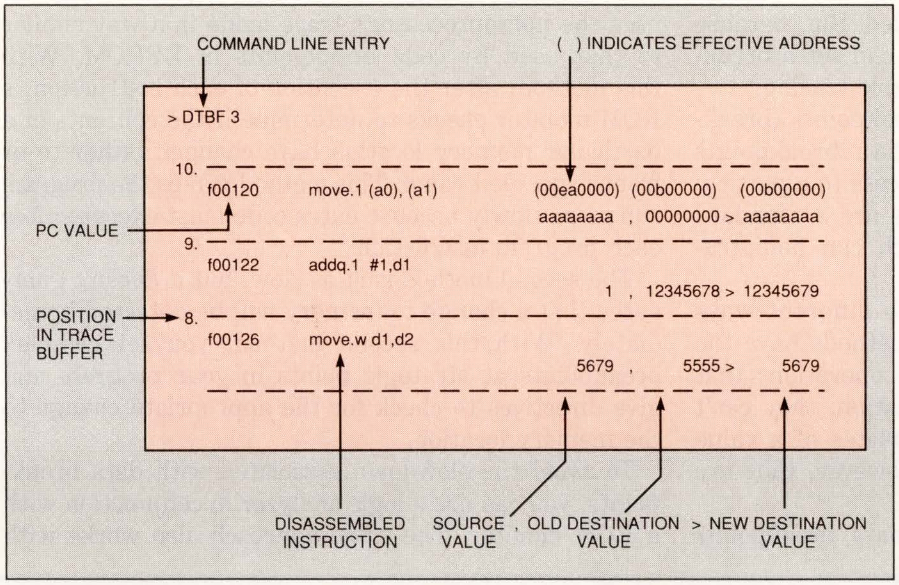


Fig 1—A ROM monitor's full-trace display can tell you not only what instructions have been executed, but also information about what those instructions have done. Here, the command DTBF 3 instructs the monitor to display a backward trace with full data movement for three instructions. For the first instruction—move.l (a0),(a1)—the display shows that the value aaaaaaaa was copied from memory at address ea0000 into memory at address b00000. In the second instruction, d1 was incremented by 1, changing its value from 12345678 to 12345679. In the third instruction, the contents of the lower-order word of d1 (5679) were copied into the lower-order word of d2, which changed from 5555 to 5679.

After storing a program counter, the ROM monitor returns control to your application program at the instruction following the one indicated by the program counter. This instruction-by-instruction sequence continues until the program encounters a breakpoint or until you interrupt the program.

Leaving the trace enabled all the time is undesirable, however. Not only does it slow down the execution of time-critical routines, but it may cause the trace buffer to be filled with information that's irrelevant to the bug you're trying to find. You may, however, be able to trace selectively. If your ROM monitor allows complex breakpoints (breakpoints followed by the execution of directives that you have previously specified), then you can set breakpoints at strategic points in the program and either enable or disable trace at each of those points.

At a breakpoint, or when you interrupt your program, you can display the accumulated trace buffer either as a simple list of program counters or as disassembled instructions. You can also instruct some ROM monitors to save additional information about the instructions they record in the trace buffer (**Fig 1**).

Often you do need more information than just which instructions have been executed. It is not enough to know, for example, that the instruction MOVE #0,(A3) has been executed if the address in A3 was incorrect to begin with. However, you can direct the ROM monitor to capture, for each instruction executed, any effective addresses and the values at those addresses before and after instruction execution. Configuring a ROM monitor for this kind of trace does exact an additional penalty in execution speed because the monitor must disassemble each instruction prior to execution. However, the monitor lets you accumulate a trace that can help to pinpoint the spot where an assignment through a bad pointer takes place.

The problem of setting breakpoints on instructions in ROM (usually EPROM) requires ROM monitors to take a different approach than for setting breakpoints in RAM. For RAM breakpoints, ROM monitors normally replace an application-program opcode with some other instruction—for example, a trap instruction. When the replacement instruction executes, an exception occurs that passes control to the monitor.

However, you can't overwrite opcodes in EPROM, so the ROM monitor again takes advantage of the target microprocessor's trace mode. But, in addition to saving the program counter value, the monitor compares it with each address previously recorded in a list of breakpoint addresses.

This technique won't work, though, if you try to set a breakpoint in an interrupt-service routine (ISR) that resides in EPROM. Interrupt processing saves the existing status register on the stack and then disables tracing. Thus, because tracing is disabled, the monitor can't check for breakpoints in EPROM after each instruction.

The solution to this problem involves forcing the trace bit to be on during execution of the ISR. One way to do this is for you to make the interrupt vector point to code in RAM that jumps to the actual ISR code in EPROM (**Fig 2**). When program control gets to this jump instruction, the old status register has

Setting breakpoints in a ROM-resident interrupt-service routine requires a little trickery.

been saved and tracing has been disabled. But, because the jump instruction is in RAM, you can set a breakpoint on it and, at the same time, enable tracing.

Sometimes, in addition to code breakpoints (breakpoints on instructions), you need data breakpoints (breakpoints that halt execution on access to a particular data location). These breakdowns are a standard feature of in-circuit emulators, which can nonintrusively monitor bus activity.

With ROM monitors, there are two different ways to simulate data breakpoints. Both methods have the limitation of checking only for write operations that change the value of the memory location; they can't detect reads, and they can't detect writes of a value that the location already contains. However, they are adequate for many purposes.

The first method for simulating data breakpoints uses the microprocessor's trace mode in a way similar to that used by code breakpoints in EPROM. With this method, after the execution of each instruction, a ROM monitor checks to determine if the contents of a particular memory location have changed either to or from a specified value. This method makes the program run more slowly because extra code must execute after each program instruction.

The second method isn't as slow, but it doesn't guarantee that a change to memory will be detected immediately. With this second method, you set complex breakpoints at strategic points in your program and give directives to check for the appropriate change to the memory location.

To avoid the slowdown associated with data breakpoints, you can use a logic analyzer in conjunction with a ROM monitor. The same approach also works with

How a ROM monitor works

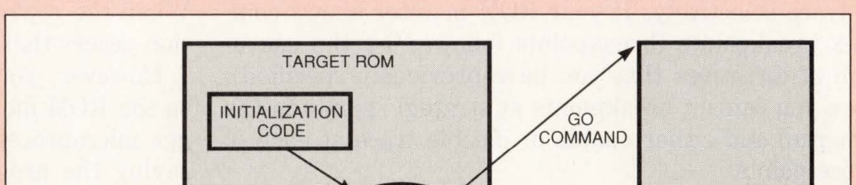
In normal use for debugging, a ROM monitor is activated upon power-up of the target board on which it resides. In most cases, you communicate with a monitor via a serial connection from a host computer or terminal. A loader routine in the monitor code lets you download application code into target RAM.

Using a ROM monitor enables you to start and stop the application program and also examine and modify values in memory and the board-resident microprocessor's registers. In addition, all ROM monitors support the two basic functions of setting breakpoints and single-stepping through code.

A ROM monitor implements code (instruction) breakpoints in RAM by removing the instructions at breakpoint locations and replacing them with trap instructions or illegal instructions. It saves the original instructions in a temporary storage area that is reserved for the ROM monitor in on-board RAM. When the application program gets to one of the trap instructions, control passes to the exception handler for that trap, which is actually an entry point into the ROM monitor.

Once entered, the ROM monitor puts back into the application code all the original instructions that were replaced with trap instructions at breakpoint locations. This action enables you to examine program memory and see your own program instructions rather than the replacement trap instructions.

To resume execution of the application program after a break-



A ROM monitor resides in ROM on a target board, but makes use of a small amount of target RAM, where it maintains a breakpoint table and a trace buffer.

tracing. The key is to use a logic analyzer's event-capturing capabilities to activate the ROM monitor. Take the logic analyzer's trigger output (a signal that is generated when a trigger condition occurs) and feed it to an interrupt on your target board's processor. An exception handler can then activate the ROM monitor, letting it display registers or other information.

In the techniques described so far, an exception—a software interrupt—has invoked the ROM monitor. But that is different than when the ROM monitor itself is interrupted. If your program uses interrupts, as many embedded-systems applications do, then much of your program's execution time might be spent in the routines that service those interrupts. And, because a ROM monitor resides on the target system, it is subject to those same interrupts.

To avoid potential problems arising from interrupts,

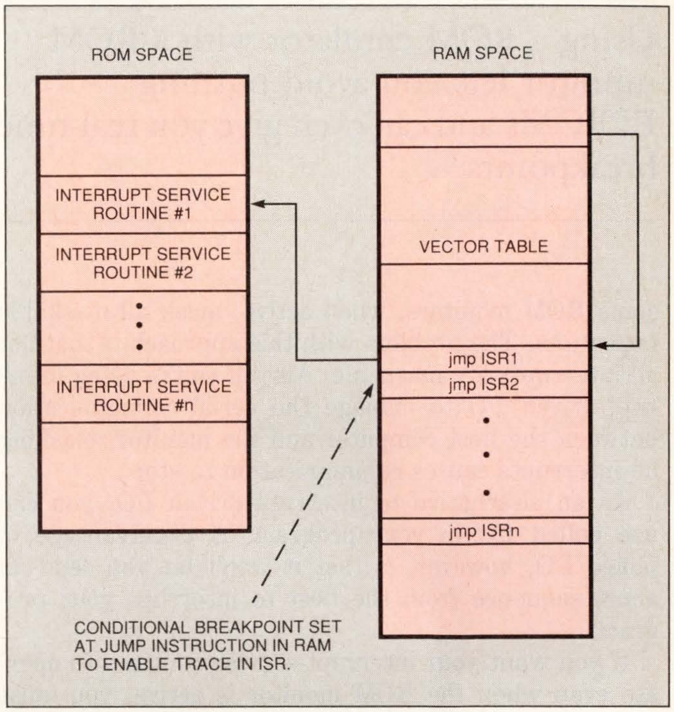


Fig 2—Setting breakpoints in ROM-resident interrupt-service routines requires a little trickery. First, make each vector in the vector table point to an entry in a jump table, which you locate in RAM. You can then set conditional breakpoints on the RAM-resident jump instructions.

point, the ROM monitor pushes the address of the breakpoint onto the stack and executes an RTE instruction, causing a return from the exception. An additional action must occur between these two steps, however, or there will be a problem: The original instructions—not the trap instructions—are in program memory, so the RTE instruction will cause execution to resume with breakpoints disabled. On the other hand, if the monitor replaces the original breakpoint-location instructions with the trap instructions before issuing the RTE, the next instruction to be executed is a trap instruction, resulting in a premature return to the monitor. (Note that the original instruction must be replaced in order to be executed.)

To avoid this dilemma, the ROM monitor sets the trace bit in the microprocessor's status register before issuing the RTE instruction. This action causes the processor to execute a single instruction and then take a trace exception, activating the ROM monitor as the exception handler. The ROM monitor then installs all the breakpoints, disables the trace bit in the status register, and finally issues the RTE instruction. The application program then proceeds until it reaches the next breakpoint.

The other basic function of ROM monitors—single stepping through the application program—also involves the trace bit. When you issue a command to single-step, the ROM monitor sets the trace bit in the status register. With this bit set, a trace exception occurs after each instruction, and the exception handler—as in the case of the trap exception handler—is an entry point into the ROM monitor.

In addition to these basic functions, most ROM monitors have additional features. Some of them might, at first, seem impossible to implement—for example, setting breakpoints in ROM where instructions are unalterable, or tracing program flow even though the monitor doesn't connect to the bus to observe processor activity. Both of these features are possible in the microprocessor's trace mode; they work, but sacrifice speed.

With ROM breakpoints, the ROM monitor puts the processor in trace mode and stores breakpoint locations in a reserved area of RAM. Because the processor is in trace mode, it activates the ROM monitor with each instruction, at which point the monitor checks the table of breakpoint locations to see if the application program has encountered a breakpoint.

Similarly, the ROM monitor builds a program-trace buffer in reserved RAM. Running in trace mode, the processor activates the monitor with each instruction; the monitor then stores the instruction location in the trace buffer.

Using a ROM emulator with a ROM monitor lets you avoid burning EPROMs and can even give you real-time breakpoints.

some ROM monitors, when active, mask all maskable interrupts. The problem with this approach is that not all interrupts are maskable. Also, if you're using interrupt-driven I/O to manage the serial communication between the host computer and the monitor, masking all interrupts causes communication to stop.

As an alternative to interrupt-driven I/O, you can use polled I/O in your program. A disadvantage of polled I/O, however, is that it won't let you send an abort sequence from the host to interrupt your program.

If you want your interrupt-service routines to operate even when the ROM monitor is active, you must ensure that your ISRs don't alter any values that the monitor may have stored in registers. If an ISR uses any registers, it must save the registers' contents upon entry to the ISR and restore them upon exit. If your ISR uses a global data register, such as A5, to access global data from your application code, then it must explicitly reassign that register after it puts the monitor's value from the register on the stack.

You can also avoid some potential problems by making sure that your application's I/O is compatible with the ROM monitor you use. If your target board has a single RS-232C port that has to be shared between the application and the ROM monitor, you can simplify the situation by making one routine manage all serial port I/O. Often, you can use the ROM monitor's I/O routine as a system call that is available to your application. If your application uses C library routines for standard I/O (*printf()*, and so forth), you need to use a version of the library that accesses the monitor's I/O system calls in the lowest-level routines. If you're using the monitor as part of an integrated tool kit (compiler, utilities, ROM monitor), look for libraries built for use with the monitor or for library source code that you can modify.

ROM emulator provides serial port

Even if your target system has no serial port, you might still be able to use a ROM monitor. Instead of a target-board port, you use the port on a ROM emulator. A ROM emulator is a hardware device with a probe that inserts into a ROM socket on your board, much as an in-circuit emulator's probe inserts into a microprocessor socket. It emulates ROM by providing RAM, a serial port, and downloading. You can use its serial port to download both your application code and the ROM monitor code. Just make sure that if a ROM emulator has your only serial port that the port is capable of bidirectional communication. You need this capability so that your host computer or terminal can receive messages from the monitor as well as send messages and download code to it.

A ROM emulator can also help you work around other target-system shortcomings. An obvious use is to "convert" plentiful on-board EPROM to emulator RAM for debugging purposes, so that you don't have to burn new EPROMs after each software change. With some ROM emulators, you can even gain real-

Some practical matters

To make use of a ROM monitor, your target system must have enough available memory space and, usually, an RS-232C port. By selecting user-configurable parameters when you install your monitor, you avoid conflicts between the monitor and your application for memory space or other resources. For example, you must specify the trap or breakpoint instructions used by the monitor.

The smallest monitors, supplied as software, occupy 2k to 3k bytes of EPROM space and a small amount of RAM. Often, these small monitors are accessible only through the interface of a source-level debugger. Larger, more capable monitors often come as EPROMs on a development board. The 332Bug monitor on Motorola's M68332EVS board, for example, uses 128k bytes of EPROM and 16k bytes of RAM. It includes a line assembler and disassembler.

You can leave a monitor in your finished product for debugging any problems that may arise after delivery. You should reconfigure the monitor, though, so that system power-up starts the application code instead of the monitor code.

Invoking the monitor in a working system normally means connecting a terminal to the system's serial port and sending an escape sequence. Alternatively, the system can activate the monitor itself—to do a memory dump, for example, or to auto-dial for field support.

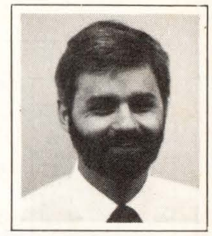
time breakpoints—that is, breakpoints without the trace-mode slowdown. For this capability, you'll need a ROM emulator that lets your target board's processor write to its RAM, thus allowing the necessary opcode overwrites.

Source-level debuggers are also useful in combination with ROM monitors. Many monitors are compatible with a debugger, and the debugger can drive the monitor much like it would drive an in-circuit emulator.

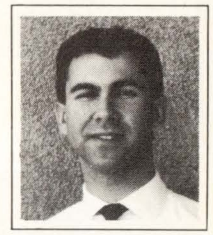
If you have very little EPROM space on your target, the "smart debugger, dumb monitor" approach can be attractive. A dumb monitor often doesn't have an interface that a user can access directly; all communication takes place via a binary protocol initiated by the source-level debugger.

If you need both source-level debugging and a complete ROM monitor, you'll prefer the "smart debugger, smart monitor" approach. You need more EPROM space for this approach, but the smart monitor lets you access it directly to perform low-level functions that are unavailable in the debugger. EDN

Authors' biographies



Peter Dawson is founder and president of Embedded Support Tools Corp (Canton, MA). Previously, he was a principal engineer at EMC Corp and a principal test engineer at the Foxboro Company. Peter holds an electrical engineering degree in instrumentation and control from the University of Hull in Great Britain.



Andy Lantz, a senior marketing engineer at Intermetrics Microsystems Software Inc (Cambridge, MA), has debugged software for a variety of applications for more than nine years. He presently provides technical support and training to software engineers designing embedded systems. Andy holds a BS degree in computer science from Brown University.

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

Adapter and software simplify interface to SCSI peripherals

Using an off-the-shelf adapter board, you can connect your computer to a SCSI bus as a host device. The board performs the functions of the low-level SCSI protocol, so instead of worrying about hardware details, you can work at the high level in software.

William C Warner, *Consultant*

Connecting your computer to a SCSI bus can give you control over many types of equipment. Almost all optical-disk drives and a growing number of magnetic-disk drives use the SCSI bus. Many peripherals such as instruments, optical-disk jukeboxes, and document scanners also have SCSI connections.

Fortunately, connecting your computer to a SCSI bus doesn't have to be an exercise in hardware design. SCSI adapter boards can handle the low-level protocol for you, letting you control SCSI devices at the high level, with software. You simply configure your adapter board and write some software. This article contains examples of routines written in C for a hypothetical adapter board that is similar to those manufactured by NCR (Dayton, OH) and Adaptec (Milpitas, CA).

The software listings in this article are available on EDN's computer bulletin board system (BBS). Phone (617) 558-4241 with modem settings 300/1200/2400,8,N,1. Access */freeware* SIG and specify *(r)ead* option followed by *(k)eyword* search for "MS #388".

SCSI connects as many as eight intelligent devices (see **box**, "SCSI basics"). IC makers offer chips that provide the interface for SCSI signals, and these chips form the core of SCSI-bus adapter boards. The boards bring about, indirectly, the connection of a computer's bus and the SCSI bus. The two buses connect only in the sense that information can pass between them; they do not connect electrically, nor does one begin to operate like the other. Off-the-shelf SCSI adapter boards are available for popular computer buses such as Intel's Multibus and the IBM PC/AT bus.

Once installed in a computer, an adapter board provides access to the SCSI bus. On the logical "bottom" of the board (**Fig 1**) is a socket for a cable through which the adapter reaches the devices on the bus. On the logical "top" of the board are control and status registers (**Figs 1** and **2**) and dual-ported memory through which software in the computer controls the adapter board. **Listing 1** shows C code (for our hypothetical board) that defines constants and static storage related to the control and status registers. (All listings begin on pg 36.)

Using onboard jumpers and switches, you configure the adaptor board so that it doesn't conflict with other

SCSI adapter boards handle the low-level protocol for you.

hardware in the host computer. This procedure establishes the base I/O address for the adapter's registers and places the adapter's dual-ported memory somewhere in the host's address space. During this procedure you also select the DMA (direct memory address) channel and the interrupt number the board will use.

SCSI devices rely on a low-level protocol to accomplish error-free transmission of data between an "initiator" and a "target" (**Ref 1**). This protocol does not depend on the content or meaning of data passing over the bus. Only the devices know what data they are exchanging; the protocol only determines which device gets to talk, to whom, when, how, and for how long.

Because ICs handle the low-level protocol and adapter boards serve as platforms for the ICs, designers connecting a computer to the SCSI bus are usually free to work on a higher level. At the higher level, devices communicate via a set of commands, such as TEST UNIT READY, READ, WRITE, and SEEK. These commands, together with the rules for their use, constitute the SCSI command protocol (**Ref 1**).

The command protocol includes standard commands

SCSI basics

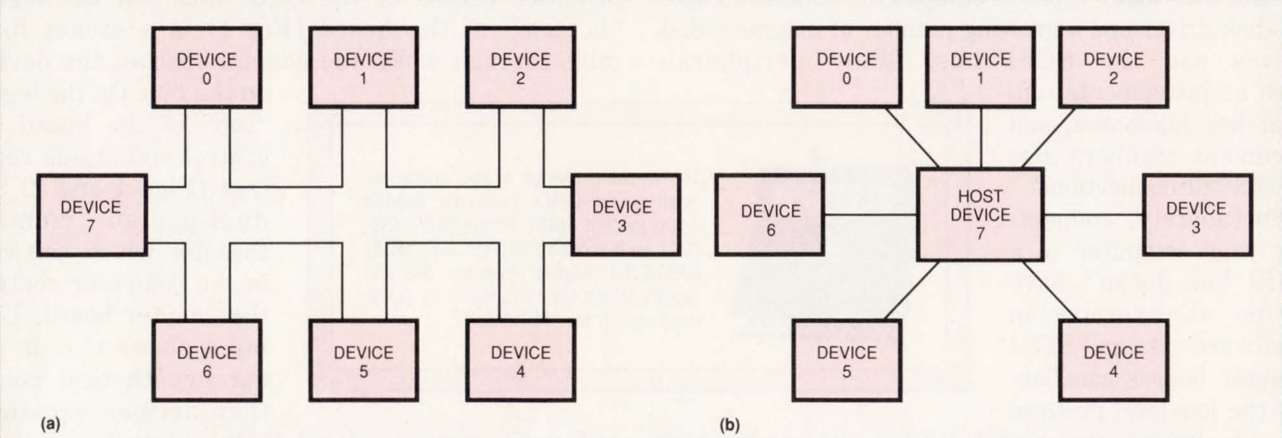
The SCSI (Small Computer System Interface) bus provides a path over which as many as eight intelligent devices can communicate. The SCSI specification defines the electrical characteristics of cables that carry signals to and from the devices, and the shape, size, and pin-out of connectors.

The SCSI specification also names the signals and defines their voltage and current levels, their timing, and the exact sequence of their interplay. In doing so, the specification lays down a low-level communications protocol. Every device on the bus must conform.

Each SCSI device has an "address." Addresses range from 0 to 7 and are similar to "drop numbers" in communications network terminology. An address gives a device a unique identity that enables it to be a sender or receiver of information. DIP switches and stab-on jumpers inside a device usually establish the device's address.

In theory, the devices at all addresses are equivalent. In practice, however, one device almost always is the boss, or "host." This device plays the role of "initiator;" the other devices are passive "targets." In most configurations, the host communicates with all the other devices, but each of them communicates only with the host. So although the SCSI bus is potentially and physically like a ring, in practice, it is logically like a star.

For additional SCSI information, read *SCSI: Understanding the Small Computer System Interface*, from NCR Corp, Dayton, OH.



The SCSI bus topology is potentially and physically like a ring in which all devices are equivalent (a). In most applications, however, one device initiates all the communication, which makes the configuration more like a star (b).

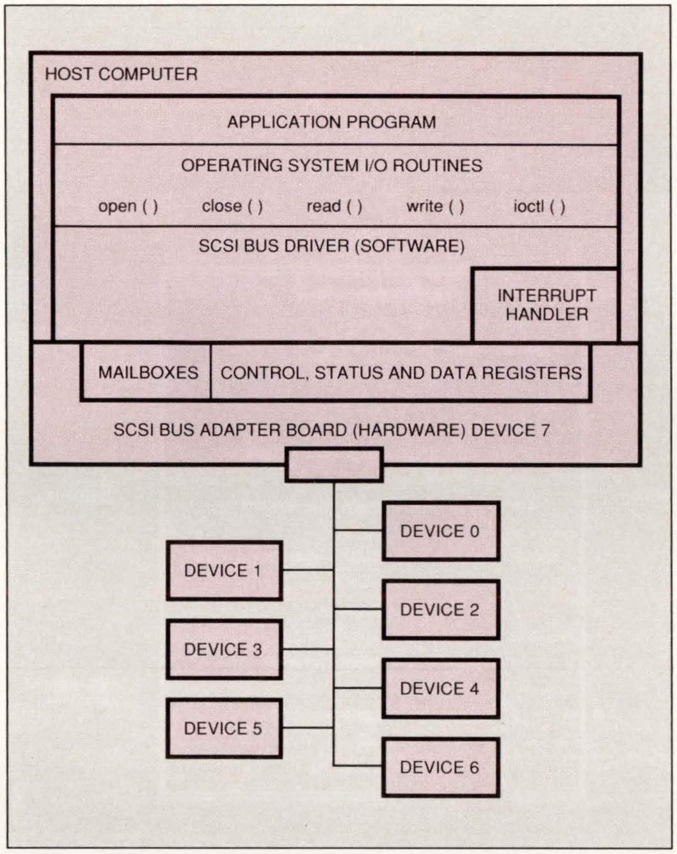


Fig 1—A SCSI adapter board connects a host computer to external SCSI devices. The computer's application program interacts with a SCSI bus driver through operating-system I/O routines. The driver controls the SCSI adapter board through onboard control, status, and data registers. When the application program directs it to, the driver sends SCSI commands through "mailboxes" in memory the driver shares with the adapter. When commands complete, the adapter interrupts the host, which invokes an interrupt handler (part of the SCSI driver) to wrap up command processing.

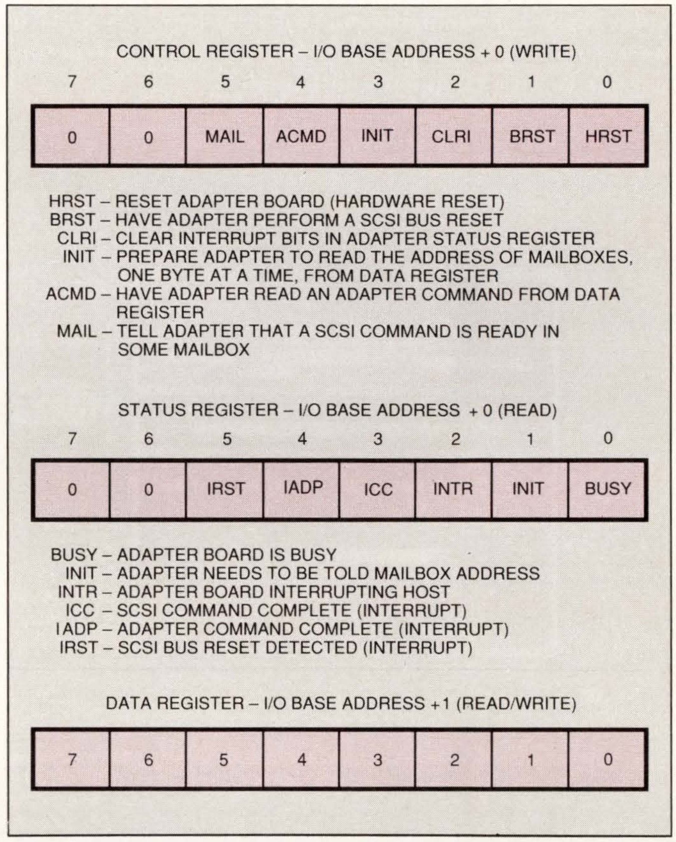


Fig 2—Control, status, and data registers, shown here for a hypothetical SCSI adapter board, let a host computer communicate with its adapter. Onboard switches or jumpers select a base address for the registers in the host's I/O space.

that every SCSI device must accommodate. The protocol also allows vendor-unique "specialty" commands that make sense only to certain kinds of equipment. For example, a SCSI-based printer might accept a specialty command to select a certain paper tray. A SCSI device will reject any inapplicable specialty command.

Every command—standard or specialty—has a command-descriptor block (CDB) of either 6 or 10 bytes **(Figs 3a, 3b,** and **3c).** Software in the host **(Fig 3d)** builds CDBs in the memory the host shares with an adapter board. The CDBs pass from the host to a target SCSI device. Fields in the CDBs tell the target device what to do.

The host sends CDBs to the adapter through "mailboxes" that are in the memory the host and adapter share. Each mailbox comprises an address field and a status field **(Fig 4a).** Typically, the host software allocates eight mailboxes **(Fig 4b)**—one for every device on the bus—and tells the adapter board the location of the mailboxes **(Listing 2).** The host can converse with more than one target at a time. (One SCSI device address belongs to the host's adapter board. The host uses the mailbox for this device to control the adapter itself with non-SCSI commands.)

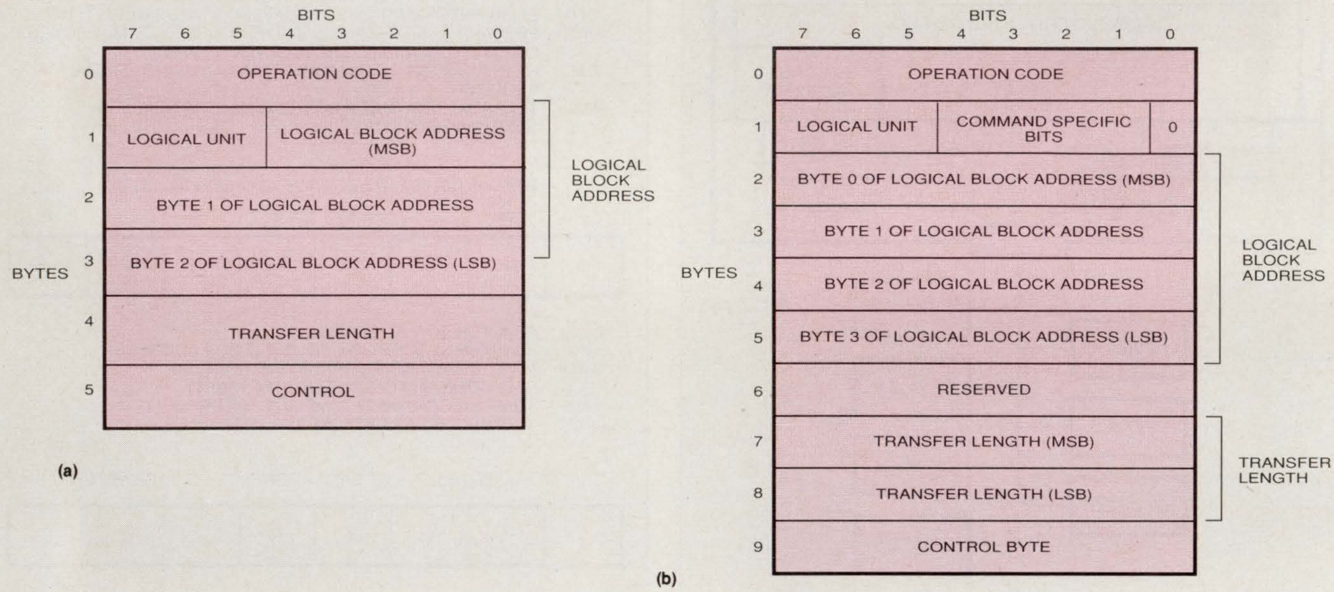
Talk to target device and adapter

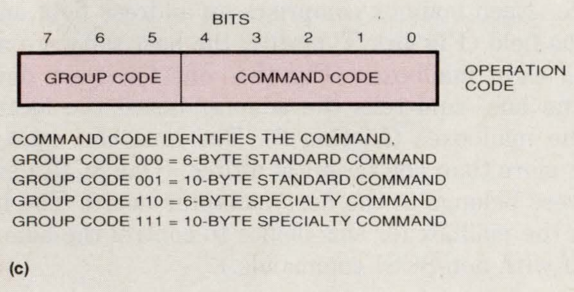
To send a command to a target device, the host software writes the address of a CDB into the address field of the target device's mailbox. Actually, to communicate with both the target device and the adapter board, the software embeds the CDB into a larger structure and writes the address of that structure into the address field. The information in the CDB is for the target device; the information in the larger struc-

Your computer's bus and the SCSI bus exchange information without connecting electrically.

ture is for the adapter. The larger structure is sometimes called a command-control structure (CCS) **(Fig 5a)**. Software for organizing memory into a structure similar to a CCS appears in **Fig 5b**.

After storing the address of the CCS (and indirectly the CDB) in the mailbox's address field, the host software stores a "ready-to-send" code in that mailbox's status field. The software then writes to the adapter's






```
/*
** Define a structure to layout memory in the form of a six-byte
** SCSI Command Block Descriptor
*/
typedef struct
{
      unsigned char OpCode;     /* Operation code                       */
      unsigned char UnitLba;    /* Unit number and LBA MSB              */
      unsigned char LbaMid;     /* LBA mid-significant byte             */
      unsigned char LbaMsb      /* LBA least-significant byte           */
      unsigned char TransLen;   /* Transfer length (blocks of bytes)    */
      unsigned char Control;    /* Control bits                         */
} CDBSIX;

/*
** Define a type to layout memory in the form of a 10-byte
** SCSI Command Block Descriptor
*/
typedef struct
{
      unsigned char  OpCode;     /* Operation code                      */
      unsigned char  UnitCmd;    /* Logical unit and command bits       */
      unsigned long  Lba;        /* Logical block address               */
      unsigned char  Reserved;   /* Reserved field                      */
      unsigned short TransLen;   /* Transfer length (blocks or bytes)   */
      unsigned char  Control;    /* Control bits                        */
} CDBTEN;
```

(d)

Fig 3—Command-descriptor blocks (CDBs) have either 6 (a) or 10 bytes (b). The only important difference between the two types is that the larger block provides an extra byte in the logical-block-address and transfer-length fields. Compared with the smaller block, the larger block can reference 256 times as many logical blocks in a mass-storage device and call for the transfer of 256 times as much data. Both types of CDBs begin with an operation code, the upper bits of which identify the command "group" (c). The lower bits of the operation code—the command code—specify the basic action required of a device that receives the command. You can define C data structures in your program to allocate memory in the form of a CDB (d).

control register to tell the adapter to check the mailboxes for outgoing commands **(Listing 3)**.

At the completion of a command, the adapter board writes a completion-status code into the target device's mailbox's status field and then interrupts the host. An interrupt handler in the host **(Listing 4)** scans the mailbox status fields until it finds a field with a completion code. After finding the mailbox for a newly completed command, the host accesses the CDB for that command. Depending on the outcome of the command, the host software may do additional processing. EDN

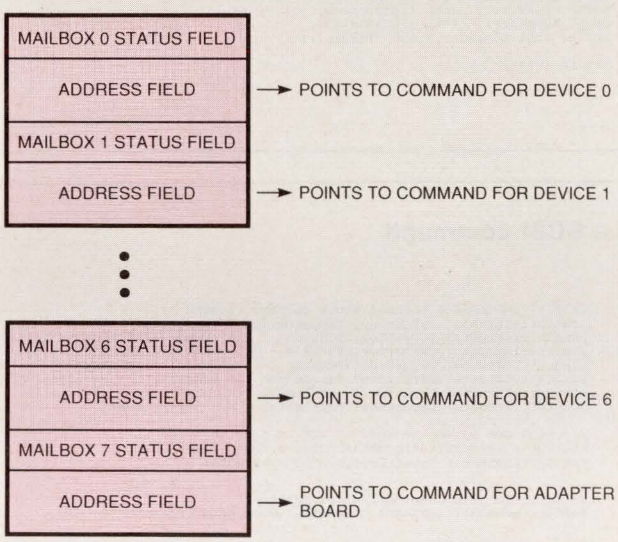
Reference

1. *Laserdrives 100 Intelligent Digital Optical Disk Drive Product Specification*, Publication No. 75115010B, Laser Magnetic Storage International, Colorado Springs, CO, 1987.

Author's biography

William C Warner is the owner of Real-Time Computer Applications, a consulting firm in Ann Arbor, MI. He has experience in software and hardware design for digitized-image processing, instrumentation, and automation. His designs have gone into CAD workstations, PC interfaces, computerized machine tools, and programmable controllers. Will has a BS in mathematics from Michigan State University (East Lansing, MI). In his spare time, he enjoys reading.

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

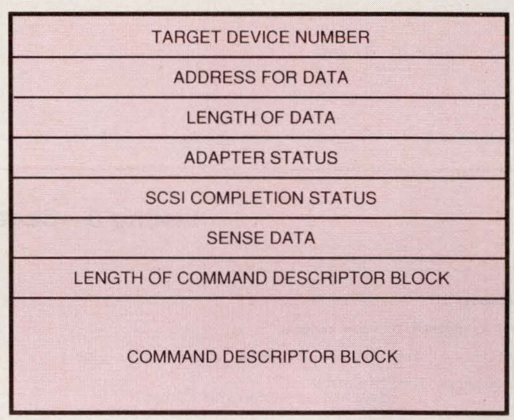


(a)

```
/*
** Define memory in the form of an adapter mailbox
*/
typedef struct
{
    char  Status;   /* set by host send to start; changed by adp done */
    char  AddrHigh; /* Most significant byte of address of cmd block  */
    char  AddrMid;  /* Mid significant byte of address of cmd block   */
    char  AddrLow;  /* Least significant byte of address of cmd block */
} MAILBOX;
```

(b)

Fig 4—Mailboxes in shared memory provide a means of communication between a SCSI adapter board and its host computer (a). The host writes a "ready-to-send" code into a mailbox status field when it has a command ready to go to a device. The adapter writes a completion-status code to the same mailbox status field when the command has completed. You can define C data structures that lay out memory in the form of mailboxes (b). Your program allocates memory for eight mailboxes and gives the base address of the memory to the adapter board so that it knows where the mailboxes are.



(a)

```
/*
** Organize memory like a Command Control Structure
*/
typedef struct
{
    unsigned char   Busy;           /* True until cmd done           */
    unsigned char   TargetDevice;   /* Device address of target      */
    unsigned char   AddrMsb;        /* Data memory most-sig byte     */
    unsigned char   AddrMid;        /* Data memory mid- sig byte     */
    unsigned char   AddrLsb;        /* Data memory least-sig byte    */
    unsigned char   DataLenMsb;     /* Data length most-sig byte     */
    unsigned char   DataLenMid;     /* Data length mid-sig byte      */
    unsigned char   DataLenLsb;     /* Data length least-sig byte    */
    unsigned char   AdpStatus;      /* Adptr stores its status here */
    unsigned char   TargetStatus;   /* SCSI cmd completion status    */
    unsigned char   SenseData[14]   /* Explain CHECK-CONDITION       */
    unsigned char   LenCDB;         /* Number of byes in CDB         */
    unsigned char   CDB[ sizeof( CDBTEN) ];  /* CBD                  */
} CCS;
```

(b)

Fig 5—A command-control structure (CCS) holds the actual information that passes between a SCSI adapter board and its host computer through a mailbox (a). The purposes of CCS fields are self-explanatory except for the sense-data field, a 14-byte block that holds SCSI "sense" data. Sense data is information about the outcome of a command when the completion status is CHECK-CONDITION (see Listing 1). A C data structure allocates memory in the form of a CCS (b).

Listing 1—Constants and static storage

```
/*
** Constants for controlling an adapter board
*/
#define ADP_CMDHRST   0x01   /* Control register bit HRST */
#define ADP_CMDBRST   0x02   /* Control register bit BRST */
#define ADP_CMDCLRI   0x04   /* Control register bit CLRI */
#define ADP_CMDINIT   0x08   /* Control register bit INIT */
#define ADP_CMDACMD   0x10   /* Control register bit ACMD */
#define ADP_CMDMAIL   0x20   /* Control register bit MAIL */

/*
** Constants for testing adapter status
*/
#define ADP_STABUSY   0x01   /* Status register bit BUSY  */
#define ADP_STAINIT   0x02   /* Status register bit INIT  */
#define ADP_STAINTR   0x04   /* Status register bit INTR  */
#define ADP_STAICC    0x08   /* Status register bit ICC   */
#define ADP_STAIADP   0x10   /* Status register bit IADP  */
#define ADP_STAIRST   0x20   /* Status register bit IRST  */

/*
** Code to signal that a mailbox has a command ready to send
*/
#define MB_CMDRDY     0xFF


/*
** SCSI command completion status.  A target device reports
** the overall outcome of a command with one of these codes
** during the STATUS phase of the low-level protocol.  The
** adapter board stores the code in a field of the Command
** Control Structure for the command for the host to analize.
*/
#define GOOD_STATUS              0x00
#define CHECK_CONDITION          0x02
#define CONDITION_MET            0x04
#define BUSY                     0x08
#define RESERVATION_CONFLICT     0x18


/*
** Store base I/O address of the adapter board and the addresses
** in host memory of the mailboxes and cmd cntrl structs
*/
static int AdpAddr;
static MAILBOX *pMailboxes;
static CCS *pCmdCntrlStructs;
```

Listing 2—Initialization code

```
/*
** AdpInit()
**
** Initialize an adapter board.
**
** C call:       int AdpInit( Addr, pMB, pCCS )
**
** Parameters:   int Addr
**                    The base I/O address of the adapter board's registers
**
**               MAILBOX *pMB
**                    The address of memory to be used as mailboxes
**
**               CCS *pCCS
**                    The address of memory to be used as cmd cntrl structs
**
** Returned:     0 for success; or
**               a negative error code
**
** This routine calls a routine to convert the address of mailboxes
** in the host's memory into the corresponding address in the adapter's
** memory then tells the adapter the address.
*/

int AdpInit( Addr, pMB, pCCS )
int      Addr;
MAILBOX *pMB;
CCS      *pCCS;
{
    int stat = 0;
    long lAdpBoxAddr;

    /* save address */
    AdpAddr          = Addr;
    pMailboxes       = pMB;
    pCmdCntrlStructs = pCCS;

    /* convert the mailboxes address into address relative adapter memory *
    lAdpBoxAddr = ConvertAddress( pMailboxes );

    /* reset adapter board; wait for not busy */
    outp( AdpAddr, ADP_CMDHRST );
    while( inp( AdpAddr) & ADP_STABUSY );

    /* tell adapter the location of mailboxes */
    outp( AdpAddr, ADP_CMDINIT );
    while( inp( AdpAddr) & ADP_STABUSY );
    outp( AdpAddr+1, (int) (lAdpBoxAddr >> 16) );
    outp( AdpAddr+1, (int) (lAdpBoxAddr >>  8) );
    outp( AdpAddr+1, (int) (lAdpBoxAddr)       );
    while( inp( AdpAddr) & ADP_STABUSY );

    return (stat);
}
```

Listing 3—Code to send a SCSI command

```
/*
** AdpSend()
**
** Send a command to some target.
**
** C call:       int AdpSend( TargetDev, pData, nLen, pCDB )
**
** Parameters:   int TargetDev
**                    SCSI device address of target
**
**               char *pData
**                    Pointer to memory as src or dst of data transfer
**
**               long lLen
**                    Numnber of bytes at *pData
**
**               void *pCDB
**                    Pointer to filled-in Command Block Descriptor
**
** Returned:     0 for success; or
**               a negative error code
**
** This routine calls a routine to "sleep" until the interrupt
** handler runs and marks the command as complete.
*/

int AdpSend( TargetDev, pData, lLen, pCDB )
int  TargetDev;
char *pData;
long lLen;
void *pCDB;
{
    int stat = 0;
    int nLenCDB;
    long lAddrAdp;

    /*
    ** Convert the data address into a three byte address relative
    ** to the start of adapter memory
    */
    lAddrAdp = ConvertAddress( pData );

    /* Fill in Command Control Block for the command */
    pCmdCntrlStructs[ TargetDev].TargetDevice = TargetDev;
    pCmdCntrlStructs[ TargetDev].AddrMsb      = (char) (lAddrAdp >> 16);
    pCmdCntrlStructs[ TargetDev].AddrMid      = (char) (lAddrAdp >>  8);
    pCmdCntrlStructs[ TargetDev].AddrLsb      = (char) (lAddrAdp);
    pCmdCntrlStructs[ TargetDev].DataLenMsb   = (char) (lLen >> 16);
    pCmdCntrlStructs[ TargetDev].DataLenMid   = (char) (lLen >>  8);
    pCmdCntrlStructs[ TargetDev].DataLenLsb   = (char) (lLen);

    /* check CDB Opcode to know if CDB is 6 or 10 bytes long */
    nLenCDB = (((CDBSIX *)pCDB)->OpCode & 0x20) ? 10 : 6;
    pCmdCntrlStructs[ TargetDev].LenCDB = nLenCDB;

    /* copy CDB into Command Control Block */
    memcpy( pCmdCntrlStructs[ TargetDev].CDB, pCDB, nLenCDB );

    /* fill in mailbox */
    lAddrAdp = ConvertAddress( &(pCmdCntrlStructs[ TargetDev]) );
    pMailboxes[ TargetDev].AddrHigh = (char) (lAddrAdp >> 16);
    pMailboxes[ TargetDev].AddrMid  = (char) (lAddrAdp >>  8);
    pMailboxes[ TargetDev].AddrLow  = (char) (lAddrAdp);
    pMailboxes[ TargetDev].Status   = MB_CMDRDY;

    /* tell adapter to check mailboxes */
    pCmdCntrlStructs[ TargetDev].Busy = 1;
    outp( AdpAddr, ADP_CMDMAIL );

    /* wait for command to complete */
    while( pCmdCntrlStructs[ TargetDev].Busy  )
        Sleep( &(pCmdCntrlStructs[ TargetDev].Busy) );

    /*
    ** Process outcome:  check the Cmd Cntrl Struct fields
    ** AdpStatus, TargetStatus, and SenseData[] to find out
    ** the outcome of the command, and respond accordingly.
    */

    return (stat);
}
```

Listing 4—Interrupt handler

```
/*
** AdpIntr()
**
** Service an interrupt from a SCSI adapter board.
**
** C call:       int AdpIntr()
**
** Parameters:   none
**
** Returned:     1 if interrupt serviced; or
**               0 if not
**
** This routine runs in response to an interrupt from the adapter
** board to wrap up processsing of some SCSI command that the host
** has issued.  This listing shows only code to deal with the
** adapter board; it does not show other things that must be done
** by valid interrrupt handlérs in various operating systems.
*/

int AdpIntr()
{
    int dev;

    /* bail out if adapter board not requesting interrupt service */
    if ( !(inp( AdpAddr) & ADP_STAINTR) )
        return (0);

    /* clear interrupt bit in adapter status register */
    outp( AdpAddr, ADP_CMDCLRI );

    /* scan mailboxes for completion status codes */
    for (dev = 0; dev < 8; dev++)
    {
       /* look for mailbox status non-zero and non-command-ready code */
       if((pMailboxes[dev].Status != 0)&&(pMailboxes[dev].Status != MB_CMDR
           break;
    }

    /* if found a complete command, wake up the process waiting for it */
    if ( dev < 8 )
    {
        pCmdCntrlStructs[ dev].Busy = 0;
        Wakeup( &(pCmdCntrlStructs[ dev].Busy) );
    }

    return (1);
}
```

Memories, ASICs, and Logic ICs Deliver High-End Performance.

For high-end workstation and PC applications, Oki offers a range of ICs with the powerful performance features your high-level board designs demand.

1-Meg Based VRAMs. Oki's high-bandwidth video RAMs enable the up-front performance required for high-resolution graphic applications. Features include dual port memory and fast access times.

0.8μm Gate Arrays. Manufactured on our volume 4-Mb line, Oki's SOGs offer exceptional benefits: high-speed logic and I/O performance, high-density macrofunctions, high pin count packages, and more.

Field Memory. There's no better solution for a frame grabber design than Oki's high-performing 1-Mb serial memory. Features include an internal self-refresh control circuit, making this device appear fully static to the user.

Speech Synthesis. For high-quality performance you can hear, no one matches Oki's RealVoice™ speech synthesizers. With on-chip filter and D/A, these chips reduce design time and IC count while increasing system reliability.

16-Bit MCU. Oki's nX family of fast MCUs combines a three-program instruction pre-fetch queue to lower overall CPU cycle time down to 200 ns. Features include a variety of I/O options plus 16K of 16-bit word ROM and 512 bytes of RAM.

Start packing more performance into your system with Oki ICs. Call 1-800-OKI-6388 for the details.

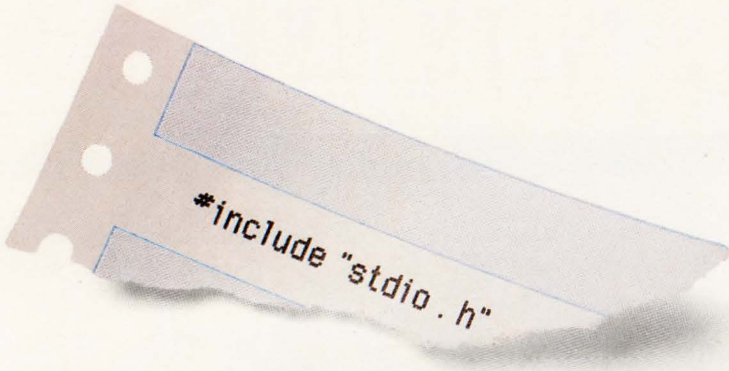
Oki High-Performance ICs

Part Number	*Description*
MSM514252	High-bandwidth, 262,144 x 4-bit VRAM
MSM514221A	262,263-word x 4-bit, 1-Mb serial memory with self-refresh control circuit
MSM10S0000	0.8μm SOGs, true 82xx, UARTs, memories, standard 24ma drive, 300ps, >500MHz logic
MSM6388	Solid-state recorder/IM serial register I/F
MSM67620	16-bit MCU with 16KB ROM, 512B RAM, 56 I/Os, 3 x 16-bit timers, 2 x 8-bit timers

TRANSFORMING TECHNOLOGY INTO CUSTOMER SOLUTIONS

RealVoice is a trademark of Oki Semiconductor.

CADRE THE SHORTEST DISTANCE BETWEEN PROMISE AND PRODUCT CADRE THE SHORTEST DISTANCE BETWEEN PROMISE AND PRODUCT CADRE THE SHORTEST DISTANCE BETWEEN PROMISE AND PRODUCT



If this is the starting line, you've already lost the race.

The race to develop software products can be brutal. Especially when the finish line keeps moving in the wrong direction.

If you're jumping into coding first thing, you probably think you're saving time on the front end. But the fact is, 7 out of 10 lines of code written *never get used.* Which means you're in for a lot of expensive and unnecessary rework.

Unnecessary because Cadre can get you on the right track from the word "go." And keep you on track throughout the entire process.

For starters, it helps to know where you're headed before you start spending money and time. Cadre's analysis toolset lets you determine and clearly document what it is you're going to build. This gives you better control over the partitioning required for complex projects, faster response to changing requirements, and improved communications and consistency down the line.

Once you have a clear idea of where you're headed, Cadre's unique simulation tool lets you test your system's behavior and estimate performance before you start to implement. So you can address misunderstandings early on, when they're easier to fix.

Cadre's design tools then help ensure that when you get into coding, you'll get it right the first time. Cadre gives the software development team the ability to see and agree on the architecture and to "walk through" the system to resolve any ambiguities up front. So instead of winding up with discrete "islands" of code that don't hang together, you arrive at an integrated system that meets your requirements.

Now that everything else is in order, it's still not too late to start writing code. Here's where Cadre's source builders step in and translate your design into actual lines of code. They automate the tedious parts, eliminate errors, and free you to focus on the creative parts. So your final, compilable code is of a much higher quality.

These are just a few of the benefits made possible by Cadre's *Unified* CASE® strategy. Cadre eases your development process with an open architecture that integrates the best tools — yours, ours, or anybody's — into your unique environment. The philosophy is simple: to minimize the distance between what you promise and what you deliver.

Take a giant step toward the finish line. Call Cadre at 1-401-351-CASE or 1-800-743-CASE today.

CADRE

THE SHORTEST DISTANCE BETWEEN PROMISE AND PRODUCT CADRE THE SHORTEST DISTANCE BETWEEN PROMISE AND PRODUCT CADRE THE SHORTEST DISTANCE BETWEEN PROMISE AND PRODUCT CADRE

PDL processors ease transition from CASE to coding

Graphics-based CASE tools can help you design software, but they're not very good for implementing a design in source code. Program-design languages (PDLs) can ease the design-to-code transition, and PDL processors can help ensure that the source code you implement is correct.

Harold Hawley and Michael Capuano,
Softsmith Inc

Graphics-based software-design tools found in computer-aided software-engineering (CASE) systems have grown more popular over the past few years, helping many software engineers with structured software design. However, today's graphics-based software-design tools are not really complete software-development tools; they don't provide an optimal means of translating the graphical design into program source code.

Program-design languages (PDLs) can help in the design-to-code translation, but they don't ensure correct design implementation. A PDL processor running on your computer can analyze your PDL pseudocode to help prevent source-code implementation errors. Without a PDL processor, you either have to skip this phase—inviting implementation errors—or do a lot of tedious and time-consuming manual checking that will be prone to error.

The main problem with today's graphics-based software-design tools is that they have an adverse effect on the human cognitive process during a project's implementation phase. Understanding why requires a basic understanding of the tools' methodology.

Most graphics-based software-design tools follow (and rigidly enforce) a structured analysis/structured design (SA/SD) methodology. They functionally and hierarchically decompose a software application via data-flow diagrams (DFDs), control flow diagrams (CFDs), and structure charts.

The SA/SD methodology for a typical DFD uses circles (bubbles) to represent software functions. At the hierarchy's highest level—the context diagram (**Fig 1**)—a single bubble represents the entire software application. All software functions performed by the application are represented in aggregate by this single bubble.

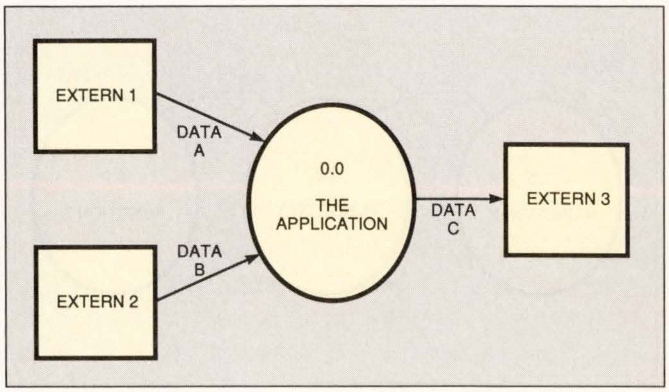


Fig 1—In a context diagram, a single bubble represents an entire software application. External interfaces to the application appear as boxes.

CASE tools fall short in translating software designs into working source code.

The external interfaces to the application being designed also appear on this top-level diagram. Labeled arrows represent the exchange of data between the application and the outside world.

The design proceeds in a hierarchical, top-down fashion. The designer decomposes the single bubble of the context diagram to arrive at a first-level DFD, which typically contains from three to seven major functions. As in the top-level DFD, these functions appear as labeled bubbles **(Fig 2).** Also as before, labeled arrows represent the flow of data—external data between functions and external entities, plus internal data between different functions.

In this first-level DFD, each bubble still represents an aggregate group of related, high-level functions. In continuing the functional decomposition to the next level, the designer again divides each function into its related sub-functions. In **Fig 2**'s first-level DFD, for example, each of the functions, X, Y, and Z, can be represented by a second-level DFD.

This functional decomposition continues until the designer has identified all functions. Ideally, the decomposition stops when each bubble in every DFD represents a single function rather than an aggregation of multiple functions. The resulting decomposed application can be represented as a tree **(Fig 3)** that typically is asymmetrical. Each unshaded box in the figure represents a single DFD; each shaded box represents an individual function within a lowest level DFD.

Diagram shows what; p-spec shows how

After determining that the processing represented by a bubble performs a single elemental function, the software designer or developer describes that function by creating a process specification (p-spec) for it. Unlike a DFD's bubbles, which graphically represent what functions an application will perform, the p-specs associated with functions specify how the application will perform those functions.

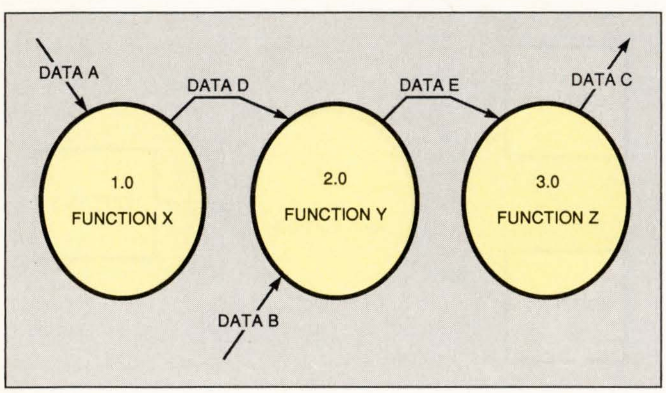


Fig 2—In a first-level data-flow diagram, each bubble represents an aggregate group of related, high-level functions.

Like data-flow analysis, the other methods assisted by graphical tools—control-flow analysis and structure-chart analysis, for example—are all concerned with the relationships among the software functions and data that compose an application. All of these methods emphasize the structure, or form, of the application, concentrating on the what and not the how.

The p-spec has a crucial role in today's graphical methodology because it is the only tool that software developers can use to describe how a given function will work. For an aircraft navigation system, for example, a software designer would graphically identify the function of computing ground speed, the data going into the computation, and the data resulting from the computation. However, the actual algorithm for how to compute ground speed would not be part of the graphical representation. It would be described in a p-spec.

The detailed information about a software design, especially the p-specs, is not easily accessible with graphics-based tools. The tools do store all the design information, including the p-specs, in their databases, but access to the data is complicated, usually requiring you to navigate manually through the hierarchically decomposed application. Shortcuts to the data may be available, but they often require some memorizing. For example, to access a p-spec without having to first display its associated DFD, you may have to have memorized the name of the p-spec file.

These access difficulties occur whether the design data are on line or on paper. Invariably, the user of a graphics-based tool must use manual methods to access data that have been generated and stored automatically.

Problem: No p-spec analysis

Another problem with graphical tools is that virtually none of them ensure that the logic represented by each p-spec is correct or complete. Most graphics-based tools require the use of p-specs, but none of them really analyze the p-specs. This lack of analysis tends to devalue the importance of the p-spec, whose purpose is to describe in detail how the corresponding function will accomplish its assigned task.

For example, graphical tools don't automatically ensure that a function uses, or even needs, its designated

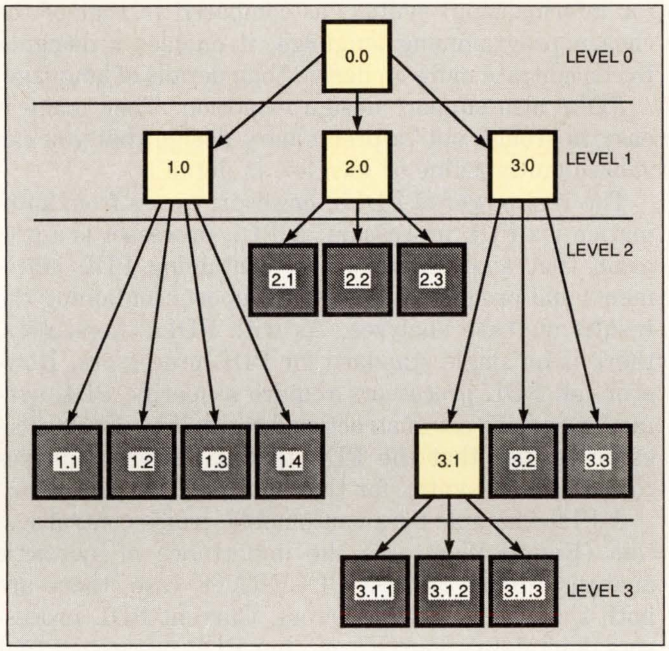


Fig 3—A tree-like structure represents an entire software application that has been broken down into lower-level functions. Unshaded boxes represent single data-flow diagrams; shaded boxes represent individual functions within the lowest level DFDs.

incoming data or that it will compute its required outputs correctly. In effect, the tools allow a software developer to rush through the task of p-spec creation. These graphical tools treat—or ignore—p-specs in a number of ways that account for developers' tendency to hurry through.

First of all, graphics-based software-design tools are naturally biased toward emphasizing their graphical interface and underutilizing text—an unfortunate imbalance. However important graphic aids may be in the early stages, text (source code) is the bottom line of software development. The p-specs (text) are just as important to this process as the bubbles, data flows, control flows, and other graphical elements.

In addition, graphical tools don't actually use p-specs, except to denote that the functions corresponding to the p-specs have been fully decomposed. By failing to perform a meaningful analysis, the tools essentially demote p-specs to markers that say only "The programmer was here."

Graphical tools also don't ensure that the logic represented in the collection of p-specs is related to the functionally decomposed software application. In other words, the tool user has no way, other than tedious manual checking, of determining whether p-specs' contents are complete, correct, or even relevant to the design.

This problem is related to the previous problem, concerning the analysis of individual p-specs, but has a broader scope. It addresses the relationships among functions across the entire design, as well as the relationship of the p-specs to the graphics-based design as a whole.

Today's graphics-based software-design tools generally do not cross-reference the contents of p-specs to the components of the graphical design, nor do they cross-reference one p-spec to another. For example, they provide no automated checking to ensure that a function called in the logic of one p-spec is correctly identified or that the function is defined either elsewhere in that p-spec or in another p-spec. Once again, this lack of capability implicitly devalues the p-spec to the detriment of the overall design.

Another failure of graphical tools is their inability to assist large-project software developers in discovering where duplication of effort may exist. In practice, a graphical design is almost never carried to the point of identifying every last function that needs to be coded. The lowest level utility functions especially are unlikely to be identified in bubbles, so you have no way to check them for duplication.

The impact on cognitive processes

All of these problems with graphical tools have a negative impact on human cognitive processes. Consequently, understanding the design of any significant system and correctly translating the design into source code—already a nontrivial task—becomes even harder. Problems with graphical tools can affect not only cognitive processes, but also productivity.

Cumbersome access to design data forces software developers to spend time and mental energy accessing information, when what they really want and need is to use the information. The net result is a diversion of concentration and a disruption of thought processes.

In addition, the lack of automated support to verify individual p-specs forces software developers to perform detailed manual checks. This process is neither interesting nor rewarding, and although it is necessary, it invites procrastination and oversight. It is a task well suited to automation, and one that—when performed manually—is rarely done well enough to enhance design quality. The lack of an automated method to ensure that p-specs relate to the graphics-based design and to each other has the same general

A PDL processor analyzes p-specs to help ensure that plans for implementing functions in source code are correct.

effect. Checking these relations is as problematic as verifying the p-specs themselves.

All of these problems result in compromising p-spec quality because the graphical tools actually discourage the cognitive process so far as p-specs are concerned. They devalue p-specs by not using them. If the tools used and analyzed p-specs, software developers would have a cognitive aid, rather than a hindrance, for creating quality p-specs.

Unfortunately, the cumulative effect on a software developer's cognitive process is greater than you might expect when considering each problem individually. The net result is that, in order to understand a design, the user of graphical tools is forced to divert time and mental energy away from the task of translating a design into working code. What a designer needs is a way to extend current design automation to bridge the gap between a graphical design and correct source code.

Problems solved by PDL processor

A text-based software-development automation tool known as a program-design language (PDL) processor can help fill that need. (When used in conjunction with today's graphical design tools, the tool might better be described as a p-spec analyzer. The terms "program-design language (PDL)," "pseudocode," and "structured English" are all synonymous. PDL is a generic term for a class of design languages; it does not refer to any specific language or product.)

All PDLs provide a structured context for specifying how a function will perform its assigned task. For example, all have basic structured constructs such as IF . . . ELSE . . . ELSEIF . . . ENDIF, and BEGIN . . . END. What distinguishes one form of PDL from another is the specific set of constructs. The desired set of constructs varies from one project to another, depending on factors such as the programming language to be used, internal company standards, and designers' and managers' personal preferences.

The lack of a single standard for PDLs is actually an advantage; it allows you to choose a standard that meets your project's specific objectives. For example, if you use a PDL that has the same basic structured constructs as your chosen programming language, the transition from correct pseudocode to correct source code will be significantly simpler.

The use of pseudocode is itself a distinct advantage. A PDL provides structure without requiring rigid adherence to detailed syntax requirements; by relaxing (or informalizing) syntax, as compared to that of the chosen programming language, it enables a designer to concentrate more on design than details of language.

PDLs also support design evolution. They make it easy to "rough out" a preliminary design that you can subsequently refine or increase in detail.

The real power of PDLs, however, comes from automation via PDL processors. A PDL processor is a program that analyzes text files containing PDL statements and produces a variety of reports containing the results of these analyses. As with PDLs themselves, there is no single standard for PDL processors. However, all PDL processors require a specific PDL syntax—a specific set of structured constructs. Each processor verifies that the PDL text it analyzes adheres to the correct syntax for that specific PDL processor.

A PDL example for an automobile cruise-control system **(Fig 4)** illustrates the importance of syntactic analysis. In the SET/ACCELERATE case, there are both syntactic and logic errors. Current PDL processors don't detect logic errors, but PDL processors that detect and report syntactic errors draw the designer's attention to the section of incorrect logic. In analyzing the text of **Fig 4,** the processor reports the incorrect pairing of DO and ENDIF and makes the designer realize that either DO . . . ENDDO or IF . . . ENDIF is required.

Beyond syntax analysis, some PDL processors produce alphabetic cross-references of design functions and design data items. Software designers can thus tell at a glance which functions call which other functions and which functions reference which data items. The cross-references make it easy to discover any functions that have been inadvertently declared but not called, and, with some PDL processors, to detect two or more functions that have been declared with the same name.

Some PDL processors can also prepare a comprehensive design document that consists of the syntactic analysis and associated error messages plus the function and data-item cross-references. The document may include a title page, a table of contents, and "pretty print" of the PDL code. Headers and footers; for example, "Preliminary Design" or "Release A00", can help track a design's progress and release history.

The key to using a PDL processor is the p-spec, which most graphics-based software-design tools recognize. The basic strategy is to use the p-spec as a repository for PDL statements and to use a PDL processor to analyze the statements. The exact implemen-

```
DO while engine is running (Note:  CCS = Cruise Control System)
   DO CASE of CCS event
      ON:
          IF CCS is off
             turn CCS ON (disengaged and desired speed not set)
          ENDIF
      OFF:
          turn CCS off
      BRAKE/TRANSMISSION:
          IF brake is on or transmission is not in a forward gear
             disengage CCS
          ENDIF
      RESUME:
          IF CCS is on but disengaged and desired speed is set
             IF transmission is in a forward gear and brake is off
                engage CCS
             ENDIF
          ENDIF
      SET/ACCELERATE:
          DO WHILE SET/ACCELERATE is depressed
             set desired speed to current speed
             IF CCS is not engaged
                engage CCS
             ELSE
                increase throttle
             ENDIF
          ENDIF
      OTHER:
          control throttle to maintain desired speed
   ENDDO
ENDDO
```

Fig 4—A PDL example for an automobile cruise-control system illustrates the importance of syntactic analysis. A PDL processor will detect the incorrect syntax pairing of DO and ENDIF statements under the SET/ACCELERATE case, alerting the software designer to a possible logic error.

tation varies, because different tools work with p-specs in different ways. The only difficulty is in determining how a particular PDL processor will access the p-spec.

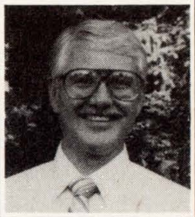
In one case, a graphics-based tool stores p-specs as separate text files that are accessible via normal file-manager functions. If your graphical tool stores each p-spec in a separate file exactly as you enter it, without attaching any additional data, then the integration task is easy. All you need to do is create a master PDL file that includes the various p-spec files in the desired order and then submit this master file to the PDL processor. The processor will read and analyze all the specified p-specs and produce a design document.

If your graphical tool modifies the p-spec data; for example, by attaching additional data such as headers, footers, or time stamps, you may need to use a "filter" program to remove the extra data. This is a relatively simple program that you can write yourself, but some graphical tools provide one for you. When you pass the p-spec files through the filter program, you create a set of files that the PDL processor analyzes.

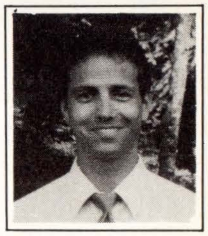
In another situation, the graphics-based tool may store p-specs in a proprietary format that isn't accessible via normal file-manager functions. This case presents a problem unless the graphical tool vendor supplies a data-extraction program or a set of data-extraction routines that can be incorporated in a user-written data-extraction program. Fortunately, given the movement toward more integrated development tools, vendors with proprietary data bases are beginning to provide this extraction capability at a reasonable cost.

Assuming, then, that data extraction is possible, the integration task is relatively easy. You use the vendor-supplied extraction capability to retrieve the p-spec data from the proprietary database, save the data in ordinary text files, and use a PDL processor for analysis. Together, the PDL processor and graphics-based design tools approach a complete software-development environment. **EDN**

Authors' biographies



Harold Hawley is president of Softsmith Inc of Bellevue, WA. He holds a BA degree in mathematics from Graceland College, an MS in applied math from the University of Colorado, and an MS in computer science from San Jose State University. Hal's hobbies include camping, tennis, boating, and theater.



Michael Capuano is vice president of Softsmith. A graduate of the University of Washington with a BS degree in mechanical engineering, he is responsible for marketing and engineering management. His spare-time interests include wind surfing, racquetball, theater, and real estate.

Article Interest Quotient (Circle One)
High 497 Medium 498 Low 499

Demand quality when purchasing software packages

Engineers should insist on the same level of quality from software that they expect from hardware. If a software package doesn't meet its specs, send it back and demand a refund.

Wayne A Gutschick, *Minc*

Quality and software—now *there* are two words that don't often go together. Why is it that we are more tolerant of poor quality in software than we are of poor quality in hardware? An engineer will pay $2500 for an oscilloscope, and if that oscilloscope doesn't meet its specifications or has a bow in the trace, the engineer would immediately send it back. Engineers would never think of using an oscilloscope that didn't meet spec.

However, the very same engineer who pays $2500 for software is willing to put up with it even if it's full of bugs! What's more, with the oscilloscope the engineer received a fully paid 1-year warranty; but with the software, the engineer gets to pay an annual maintenance fee so that maybe in the future the software bugs will be fixed and the engineer might get what he or she originally paid for. A maintenance fee is supposed to be for on-going support and technical assistance, rather than for fixing problems. I don't understand why management is willing to foot the bill.

I often hear people say that software is too complex to test and that thorough testing is impractical. I refuse to accept that premise. If IC manufacturers can figure out how to test complex 32-bit microprocessors adequately, software manufacturers should be able to figure out how to test their software to ensure quality.

I'd be a rich man if I had a dollar for every time I heard a software user say: "I'm not going to switch to a new brand because I already know where all the bugs are in my existing brand and how to work around them."

Who's at fault? I submit that it's the software purchasers. We have allowed manufacturers to set our expectations of software quality far too low. If our expectations were higher and we sent buggy software back to the manufacturer demanding our money back, I suggest that the quality problems would be addressed. The IC industry responded to pressure to improve quality in the early 1980s, and the same thing can happen in the software industry in the early 1990s.

As software purchasers and users, how can you and I judge the quality of a package before we purchase it? I have 20 suggestions:

- Talk to others who have purchased the product. Ask them about their experiences. If a substitute product were available, would they purchase the original product or give the substitute a chance?

- Ask the seller to provide you with a demonstration copy for evaluation. Don't settle for anything less than the actual product—in

Treat the purchase of software just like you would a hardware—or any other—purchase.

other words, no demo disk. You'd do the same if you were buying that oscilloscope, wouldn't you?

Ask for a specification sheet. That's right, a plain old boring spec sheet that tells you the software's performance limits. Again—just like the oscilloscope—if there's no spec sheet, how can the manufacturer verify the product's quality?

Use the product and call the manufacturer's support line for assistance. How long does it take for the phone to be answered? How quickly can you get in touch with someone who can actually help you with your problems? Did the support personnel actually help you with your problem, or did they simply direct you to a section of the manual? Were the support personnel helpful in finding a solution to your problem?

Ask the support people about their backgrounds. Are they engineers with experience in industry and with the software in question? Are they able to understand what you are doing with their software and why?

Within a reasonable period after your question has been answered, does someone from the support organization follow up to see if your problem has been solved? Are you satisfied with the answers they have given you? Do the support people care?

Call the manufacturer and ask to speak to the person in charge of software quality. The operator's not knowing whom to direct your call to is probably a good indication of the emphasis the firm places on quality. If you can reach someone responsible for quality, ask what the company has done to ensure the quality of the software package.

Ask about the manufacturer's test suites for the software. Does the manufacturer even have a test suite? How are the test suites kept current? How much of the software does the test suite test? Does the test suite run on all the platforms the manufacturer supports?

Does the manufacturer have engineers dedicated to quality assurance, or do development engineers solely dictate the quality of the released software? If there are quality-assurance engineers, are these people knowledgeable about the product? Are they testing the product in the environment for which it was intended, or are they simply pushing buttons to see if the software functions?

Can the manufacturer provide references of companies that have used the software and have complimented its quality or the time savings they realized by using it?

If the product runs on multiple platforms, how does the manufacturer support these platforms? What are the limitations of the product on each platform? What level of testing is performed on the various platforms? How does the software handle operating-system inconsistencies? Has the product been tested not only on the suggested platform but also on clone hardware?

Have other vendors chosen the software manufacturer's products for integration and/or resale on an OEM basis? Have many industry-leading companies standardized on the product? The amount of time that any one engineer can spend evaluating software is obviously limited. Prior to entering an OEM agreement, however, these larger companies have most likely completed an exhaustive evaluation.

Does the company have any patents issued or pending on the product? Patents indicate that the product takes a unique and novel approach to the problem—one that most likely will result in time savings and productivity improvements for the engineers in your company.

Are the company and its people actively involved in industry trade groups? Involvement—particularly active involvement—in such groups demonstrates a level of commitment to the success and promotion of not only the manufacturer, but also the industry in general.

What is the company's primary business? Many companies sell software as a "second line" or as something to augment the sale of hardware. These companies may not be as committed to the quality of their software as a company that strictly dedicates itself to the development and sale of software.

Is the company willing to guarantee the software in writing? If you're not satisfied with the purchase for any reason, is the company willing to refund your purchase price within a reasonable time frame? If not, why not?

Does the manufacturer have a bug-reporting-and-tracking system? When you, the customer, call up with a defect in the software, what is the company's response? How are bugs reported and tracked? Does the company assure that defects will be fixed in a subsequent release? What is the manufacturer's commitment to fixing problems?

Is there a bug-weighting system that establishes the severity of each detected defect? Who determines the severity of the defects? Is the severity of a defect based on customer input or does someone in R&D assess each defect? How does the company respond to bugs of different levels of severity?

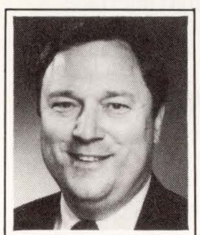
How are products and upgrades released? Is there a formal sign-off process prior to product shipment? Who must sign off before a product can be released to customers?

Are alpha and beta testing done outside the manufacturer's facility? If so, how does the manufacturer select these sites? How many test sites does the manufacturer maintain? Is the feedback from the test sites incorporated into subsequent releases? What are the criteria for your company to become a test site for future products or releases?

Do your homework, especially if the software package you're considering is a design-automation tool. You may not uncover serious defects in design software until your design is completed and breadboarded or—worse yet—in production. Also, be careful of terminology. The same term or word may have different meanings for different software manufacturers. Terminology in the software industry has few agreed-upon meanings, and customers can often be misled.

Software is a buyer-beware market. Treat the purchase of software just like you would a hardware (or any other) purchase. Doing your homework up front will be well worth your time and effort in the long run. EDN

Author's biography



Wayne A Gutschick is the president of Minc (Colorado Springs, CO) and has been with the company four years. He received his BSEE from Purdue University (West Lafayette, IN) and his MBA from Lake Forest College (Lake Forest, IL). In his spare time, Wayne enjoys handball, racquetball, skiing, and hiking. He is also active in church activities.

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

EDITED BY ANNE WATSON SWAGER

C routine efficiently searches list

Dave Bushong
Wang Labs Inc, Lowell, MA

The classic method for a linear search of an unordered list, such as the list shown in **Listing 1** (which displays a text message when passed a numerical value), is to start at the first entry, compare the entry to the given value, and return a pointer to the corresponding string if the two numbers match. If the two numbers don't match, the standard program (**Listing 2**) checks the next entry in the list, and so on, until it reaches the end of the list. This method alternately compares the current message number against the desired number and also the current index against the bounds of the array. It's necessary for the program to check the bounds because the message number may not be in the list. Most C compilers will repeatedly load registers from memory to perform these operations, which results in less-than-efficient code.

Listing 1—Unordered list example

```
struct message_pairs {
      int message_number;
      char *ascii_text;
};

struct message_pairs list[] = {
      1,    "Please insert disk",
      7,    "Drive error encountered",
      33,   "Press any key to continue",
                /* and so on */
      999,  "Unknown error occurred"
};

#define LAST_MSG \
((sizeof(list)/sizeof(struct message_pairs))-1)
```

Listing 2—Standard search routine

```
char *search(int msg_code)
{
      int i;

      for(i=0; i<=LAST_MSG; i++)
            if(list[i].message_number == msg_code)
                  return (list[i].ascii_text);

      /* here if no match */

      return ("No message available");
}
```

Listing 3—Improved search routine

```
char *search(int msg_code)
{
      int i= -1;

      list[LAST_MSG].message_number = msg_code;

      while(list[++i].message_number != msg_code)
            ;
      return (list[i].ascii_text);
}
```

A superior algorithm (**Listing 3**) places the number you're looking for into the last position in the list, which guarantees a match when you linearly search the list. In addition to generating more efficient code, this method eliminates the possibility that your array search will exceed the data in the array.

EDN BBS /DI_SIG #976 **EDN**

To Vote For This Design, Circle No. 746

DSP µP implements linear prediction

Vladimir Bochev
Center of Informatics and Computer Technology, Sofia, Bulgaria

Listing 1 (which you can obtain from the EDN BBS (617-558-4241,300/1200/2400,8,N,1—from main menu, enter (s)ig, <s/di_sig>, rk977)) contains two versions of the Shur-Leroux-Geuguen (SLG) method of solving the autocorrelation normal equations necessary for linear predictive decoding. Linear predictive coding is a fairly convenient and robust technique for modeling human speech processes. Compared to the FFT, the linear-predictive method has some advantages because of its shorter feature vector and its fit to the model of the speech-production mechanism. Linear predictive coding relies on the fact that it's possible to predict the next value of the speech signal—the next speech sample coming from an A/D converter, for example—using a linear combination of a number of previous values.

The first part of **Listing 1** is a C prototype for the SLG algorithm. An assembly-language implementation for the TMS32010 follows. At a sampling rate of 10 kHz and 128 samples for the autocorrelation, a tenth-order model computes in less than 4 msec on a custom-designed DSP board. You can run the code on any TMS320C1X or TMS320C2X without modification.
EDN BBS /DI_SIG #977 EDN

To Vote For This Design, Circle No. 747

Listing 1—TMS32010 implementation of linear predictive algorithm

```
Solver based on a "C" prototype

/* r[i] is the correlation vector array, uary[i] and vary[i] are
   working arrays for the algorithm. pvary is a pointer which helps
   for an easier and faster implementation. k[i] is the solutions
   vector, p is the order of the predictor  */
.
.
.
pvary = vary;
for(i = 0; i <= 10; i++) uary[i] = vary[i] = r[i];
/* the order of the predictor is fixed to ten for this example */
        for(i = 1; i <= p; i++) {
                pvary++;
                k[i] = *pvary/uary[0];
        for(j = 0; j <= p-i; j++) {
                        temp = uary[j];
                        uary[j] -= k[i] * pvary[j];
                        pvary[j] -= k[i] * temp;
                            }
                         }
.
.
.

now follows the tms32010 code for implementing the SLG LPC. Variables
have the same meaning as in the "C" prototype. FACTOR is a temporay
variable and PARCOR0 is the starting address of the result vector.

             LARP   AR0           ;WHICH POINTER TO START WITH
             LARK   AR0,VARY10    ;AR0 POINTS TO THE TOP OF VARY
             LARK   AR1,UARY10    ;AR1 POINTS TO THE TOP OF UARY
FILLIN       TBLR   *,AR1         ;DOUBLE TRANSFER. ACCUMULATOR
                                  ;IS ASSUMED TO HOLD THE ADDRESS
                                  ;IN PROGRAM MEMORY WHERE THE
                                  ;AOTOCORELLATION SAMPLES ARE
                                  ;STORED BY THE PREVIOUS STEP
             TBLR   *-,AR0 ;THIS IS THE SAME AS
             SUBS   ONE           ;VARY[I]=UARY[I]=R[I]
                                  ;ONE AND ZEROC ARE DATA MEMORY
                                  ;LOADED ON INITIALIZING WITH
                                  ;1 AND 0 RESPECTIVELY
             BANZ   FILLIN ;IN THE "C" PROTOTYPE


             LDPK   ZEROC         ;SELECT ZERO DATA MEMORY PAGE

; SHUR-LEROUX-GUEGUEN PARCOR ALGORITHM IMPLEMENTED WITH LOOPED CODE

             LACK   VARY0         ;INITIALIZE ADDRESS OF PVARY
             SACL   PVARY         ;LIKE IN THE "C" PROTOTYPE
             LACK   ONEC          ;ONEC IS THE CONSTANT 1
             SACL   I             ;INITIALIZE FIRST LOOP INDEX
             LACK   PARCOR0       ;INITIALIZE PARCOR INDEX
             SACL   K

LPCLOOP      LACK   ORDER         ;THIS IS THE FOR LOOP
             SUBS   I             ;CONSTRUCT FROM THE "C"
             BLZ    OUTLPC ;PROTOTYPE
             LARP   AR0           ;MAY BE SHIFTED OUT OF THE LOOP
             LAR    AR0,PVARY     ;INIT POINTER WITH PVARY
             ZALS   UARY0         ;LOAD ACCUMULATOR WITH
                                  ;DENOMINATOR
             MAR    *+            ;UPDATE PVARY POINTER
             ADDH   *             ;GET NUMERATOR
             SAR    AR0,PVARY     ;UPDATE PVARY FOR THE NEXT PASS
             SACL   DENOM         ;DENOMINATOR ADDRESS
             SACH   NUMERA ;NUMERATOR ADDRESS
             CALL   DIVIDE ;WARNING! DIVIDE USES AR0
             LAR    AR0,K         ;LOAD POINTER WITH PARCOR
                                  ;ADDRESS
             ZALS   QUOT          ;THE QUOTIENT FROM THE DIVIDE
                                  ;ROUTINE THAT CONTAINS THE
                                  ;CURRENT K[I] COEFFICIENT
             SACL   FACTOR
             SACL   *+            ;UPDATE K POINTER
             SAR    AR0,K

             LARK   AR0,UARY0     ;LOAD CURRENT UARY ADDRESS
             LAR    AR1,PVARY     ;THE "C" PROTOTYPE
             LACK   ZEROC
             SACL   J             ;INNER LOOP INDEX
             LT     FACTOR ;T DOESN'T CHANGE OVER THE LOOP

INTRALP      LACK   ORDER         ;THE INNER FOR LOOP CONSTRUCT
             SUBS   I
             SUBS   J             ;FROM THE "C" PROTOTYPE
             BLZ    IUPDATE

             ZALS   *,AR1         ;LOAD FROM UARY AND SELECT NEXT
                                  ;POINTER
             SACL   TEMP          ;SAME AS TEMP=UARY[J] IN
             LAC    TEMP,15       ;SHIFT TO FIT THE RESULT FORMAT
             MPY    *,AR0         ;OF THE MULTIPLICATION
             SPAC                 ;SAME AS UARY[J]-=K[I]*PVARY[J]
             SACH   *+,1,AR1      ;STORE IN UARY[
             LAC    *,15          ;LOAD VALUE FROM PVARY[J]
             MPY    TEMP          ;CLOSE TO THE "C" PROTOTYPE
             SPAC                 ;PVARY[J] -= K[I]*TEMP
             SACH   *+,1,AR0      ;STORE IN PVARY[J] AND UPDATE
                                  ;POINTER
             LACK   ONEC          ;END OF INNER FOR LOOP
             ADDS   J             ;CONSTRUCT
             SACL   J
             B      INTRALP

IUPDATE      LACK   ONEC          ;END OF OUTERLOOP FOR
             ADDS   I             ;CONSTRUCT
             SACL   I
             B      LPCLOOP

OUTLPC       NOP
             .
             .
             .
DIVIDE       LARP   AR0   ;STANDART TMS32010 FRACTIONAL
             LT     NUMERA ;DIVIDE ROUTINE
             MPY    DENOM
             PAC
             SACH   TEMPSGN       ;TEMPORARY SAVE FOR SIGN
             LAC    DENOM
             ABS
             SACL   DENOM
             ZALH   NUMERA
             ABS
             LARK   AR0,14
KEEPDV       SUBC   DENOM
             BANZ   KEEPDV
             SACL   QUOT
             LAC    TEMPSGN
             BGEZ   DIVDONE
             ZAC
             SUB    QUOT
             SACL   QUOT
DIVDONE      RET
```

 CIRCLE NO. 7

Estimator generates exponential function

John Dunn
Centroid Inc, Syosset, NY

Using the estimator program in **Listing 1,** you can generate an exponential function of the form $Y = A1 + A2 * e^{-X/A3}$ from raw data that looks exponential when you plot it. A1 and A2 are constants, and A3 is τ. If you submit three values of X and Y from your empirical data to the program in the **listing,** the program will generate values for A1, A2, and A3. The program is in Hewlett-Packard Basic.

The flow chart in **Fig 1** diagrams the program's algo-

Listing 1—Exponential estimator program (HP Basic)

```
10     PRINT CHR$(12)
20     GRAPHICS OFF
30     INITIALIZE ":MEMORY,0,1",103
40     CREATE ASCII "_______:MEMORY,0,1",100
50     ASSIGN @Print TO "_______:MEMORY,0,1"
60     DIM Result$[80],A$[80]
70     GOTO 210
80 Space:   !
90     PRINT
100    OUTPUT Result$ USING 120;""
110    OUTPUT @Print;Result$
120    IMAGE K,#
130    RETURN
140 Print_it:  !
150    PRINT A$
160    OUTPUT Result$ USING 180;A$
170    OUTPUT @Print;Result$
180    IMAGE K,#
190    RETURN
200    !
210    A$="THIS PROGRAM ESTIMATES THE EQUATION OF AN EXPONENTIAL WAVEFORM"
220    GOSUB Print_it
230    A$="GIVEN THREE Y-AXIS MEASUREMENTS AND THE X-AXIS VALUES OF THOSE"
240    GOSUB Print_it
250    A$="MEASUREMENTS."
260    GOSUB Print_it
270    GOSUB Space
280    INPUT "1st Y-AXIS AND X-AXIS VALUES (Y1,X1):",Y1,X1
290    INPUT "2nd Y-AXIS AND X-AXIS VALUES (Y2,X2):",Y2,X2
300    INPUT "3rd Y-AXIS AND X-AXIS VALUES (Y3,X3):",Y3,X3
310    Xx=0
320    A3=10                  ! INITIAL VALUE FOR "Tau" OF EXPONENT
330    A3_modifier=1.E-1
340    K=(Y1-Y2)/(Y1-Y3)
350    OFF ERROR
360    ON ERROR GOTO 400
370    L=EXP(-X1/A3)-EXP(-X2/A3)
380    L=L/(EXP(-X1/A3)-EXP(-X3/A3))
390    GOTO 450
400    OFF ERROR
410    A$="REAL UNDERFLOW ENCOUNTERED. CURVE NOT CALCULABLE."
420    GOSUB Print_it
430    GOSUB Space
440    GOTO 280
450    IF K>L THEN A3=A3/(1+A3_modifier)
460    IF K<L THEN A3=A3*(1+A3_modifier)
470    Xx=Xx+1
480    IF Xx>1000 THEN GOTO 1050
490    IF ABS(K-L)<A3_modifier THEN GOTO 510
500    GOTO 360
510    IF A3_modifier<1.E-5 THEN GOTO 540
520    A3_modifier=A3_modifier/2
530    GOTO 360
540    A2=(Y1-Y3)/(EXP(-X1/A3)-EXP(-X3/A3))
550    A1=Y1-A2*EXP(-X1/A3)
560    OFF ERROR
570    PRINT USING 600;"FOR Y=",Y1,",",Y2,",  &",Y3
580    OUTPUT Result$ USING 610;"FOR Y=",Y1,",",Y2,", &",Y3
```

Listing 1—Exponential estimator program (HP Basic) *(continued)*

```
590   OUTPUT @Print;Result$
600   IMAGE K,1X,SD.4DESZZ,K,1X,SD.4DESZZ,K,1X,SD.4DESZZ
610   IMAGE K,1X,SD.4DESZZ,K,1X,SD.4DESZZ,K,1X,SD.4DESZZ,#
620   PRINT USING 650;"AT  X=",X1,",",X2,",  &",X3,":"
630   OUTPUT Result$ USING 660;"AT  X=",X1,",",X2,",  &",X3,":"
640   OUTPUT @Print;Result$
650   IMAGE K,1X,SD.4DESZZ,K,1X,SD.4DESZZ,K,1X,SD.4DESZZ,1X,K
660   IMAGE K,1X,SD.4DESZZ,K,1X,SD.4DESZZ,K,1X,SD.4DESZZ,1X,K,#
670   GOSUB Space
680   Sgn$=""
690   IF SGN(A3)=1 THEN Sgn$="-"
700   IF SGN(A3)=-1 THEN Sgn$="+"
710   PRINT USING 740;"Y =";A1;A2;"* exp(";Sgn$;"X/";ABS(A3);")"
720   OUTPUT Result$ USING 750;"Y =";A1;A2;"* exp(";Sgn$;"X/";ABS(A3);")"
730   OUTPUT @Print;Result$
740   IMAGE K,1X,SD.4DESZZ,1X,SD.4DESZZ,1X,K,K,K,D.4DESZZ,K
750   IMAGE K,1X,SD.4DESZZ,1X,SD.4DESZZ,1X,K,K,K,D.4DESZZ,K,#
760   GOSUB Space
770   Chk1=A1+A2*EXP(-X1/A3)
780   Chk2=A1+A2*EXP(-X2/A3)
790   Chk3=A1+A2*EXP(-X3/A3)
800   Pct1=(Chk1-Y1)/Y1/.01
810   Pct2=(Chk2-Y2)/Y2/.01
820   Pct2=(Chk3-Y3)/Y3/.01
830   A$="RE-CHECKING THIS EQUATION AT THE SPECIFIED X-AXIS POINTS:"
840   GOSUB Print_it
850   PRINT USING 940;"Y1 =";Y1;", Y1 check =";Chk1;"ERROR =";Pct1;"%"
860   OUTPUT Result$ USING 950;"Y1 =";Y1;", Y1 check =";Chk1;"ERROR =";Pct1;"%"
870   OUTPUT @Print;Result$
880   PRINT USING 940;"Y2 =";Y2;", Y2 check =";Chk2;"ERROR =";Pct2;"%"
890   OUTPUT Result$ USING 950;"Y2 =";Y2;", Y2 check =";Chk2;"ERROR =";Pct2;"%"
900   OUTPUT @Print;Result$
910   PRINT USING 940;"Y3 =";Y3;", Y3 check =";Chk3;"ERROR =";Pct3;"%"
920   OUTPUT Result$ USING 950;"Y3 =";Y3;", Y3 check =";Chk3;"ERROR =";Pct3;"%"
930   OUTPUT @Print;Result$
940   IMAGE K,1X,SD.4DESZZ,K,1X,SD.4DESZZ,3X,K,3D.2D,1X,K
950   IMAGE K,1X,SD.4DESZZ,K,1X,SD.4DESZZ,3X,K,3D.2D,1X,K,#
960   IF X1<>0 THEN
970   GOSUB Space
980   PRINT USING 1010;"STARTING VALUE ON Y-AXIS IS",A2+A1
990   OUTPUT Result$ USING 1020;"STARTING VALUE ON Y-AXIS IS",A2+A1
1000  OUTPUT @Print;Result$
1010  IMAGE K,1X,SD.4DESZZ
1020  IMAGE K,1X,SD.4DESZZ,#
1030  END IF
1040  GOTO 1070
1050  GOSUB Space
1060  PRINT "NOT CALCULABLE. CURVE NOT EXPONENTIAL."
1070  A$=" "
1080  GOSUB Space
1090  PRINT "COMPUTE FOR SOME OTHER X-AXIS VALUE (Y/N) ? ";
1100  INPUT A$
1110  PRINT A$
1120  IF A$="N" THEN GOTO 1260
1130  IF A$="n" THEN GOTO 1260
1140  IF A$="Y" THEN GOTO 1170
1150  IF A$="y" THEN GOTO 1170
1160  GOTO 1100
1170  GOSUB Space
1180  INPUT "X = ?",X4
1190  Y4=A1+A2*EXP(-X4/A3)
1200  PRINT USING 1230;"FOR X =";X4;"Y =";Y4
1210  OUTPUT Result$ USING 1240;"FOR X =";X4;"Y =";Y4
1220  OUTPUT @Print;Result$
1230  IMAGE K,1X,SD.4DESZZ,3X,K,1X,SD.4DESZZ
1240  IMAGE K,1X,SD.4DESZZ,3X,K,1X,SD.4DESZZ,#
1250  GOTO 1070
1260  ASSIGN @Print TO *
1270  A$=" "
1280  INPUT "HARD COPY (Y/N) ? ",A$
1290  IF A$="N" THEN GOTO 1340
1300  IF A$="n" THEN GOTO 1340
1310  IF A$="Y" THEN GOTO 1330
1320  IF A$<>"y" THEN GOTO 1270
1330  COPY "________:MEMORY,0,1" TO "/spool/________"
1340  END
```

Design Entry Blank

$100 Cash Award for all entries selected by editors. An additional $100 Cash Award for the winning design of each issue, determined by vote of readers. Additional $1500 Cash Award for annual Grand Prize Design, selected among biweekly winners by vote of editors.

To: Design Ideas Editor, EDN Magazine
Cahners Publishing Co
275 Washington St., Newton, MA 02158

I hereby submit my Design Ideas entry.

Name ______

Title ______ Phone ______

Company ______

Division (if any) ______

Street ______

City ______ State ______

Country ______ Zip ______

Design Title ______

Home Address ______

______

Social Security Number ______
(Must accompany all Design Ideas submitted by US authors)

Entry blank must accompany all entries. Design entered must be submitted exclusively to EDN, must not be patented, and must have no patent pending. Design must be original with author(s), must not have been previously published (limited-distribution house organs excepted), and must have been constructed and tested. Please submit software listings and all other computer-readable documentation on a 5¼-in. IBM PC disk.

Exclusive publishing rights remain with Cahners Publishing Co unless entry is returned to author or editor gives written permission for publication elsewhere.

In submitting my entry, I agree to abide by the rules of the Design Ideas Program.

Signed ______

Date ______

ISSUE WINNER

The winning Design Idea for the March 14, 1991 issue is entitled "One coax cable carries video and power," submitted by Jeff Kirsten and Charlie Allen of Maxim Integrated Products (Sunnyvale, CA).

The winning Design Idea for the March 28, 1991 issue is entitled "Pulse generator wrings out scopes," submitted by Jim Williams of Linear Technology Corp (Milpitas, CA).

Your vote determines this issue's winner. All designs published win $100 cash. All issue winners receive an additional $100 and become eligible for the annual $1500 Grand Prize. **Vote now**, by circling the appropriate number on the reader inquiry card.

rithm. You can obtain the **listing** from the EDN BBS. Phone (617) 558-4241 with modern settings 300/1200/2400, 8, N, 1. From the main menu, enter (s)ig, <s/di_sig>, rk947). **(EDN BBS /DI_SIG #947)** EDN

To Vote For This Design, Circle No. 750

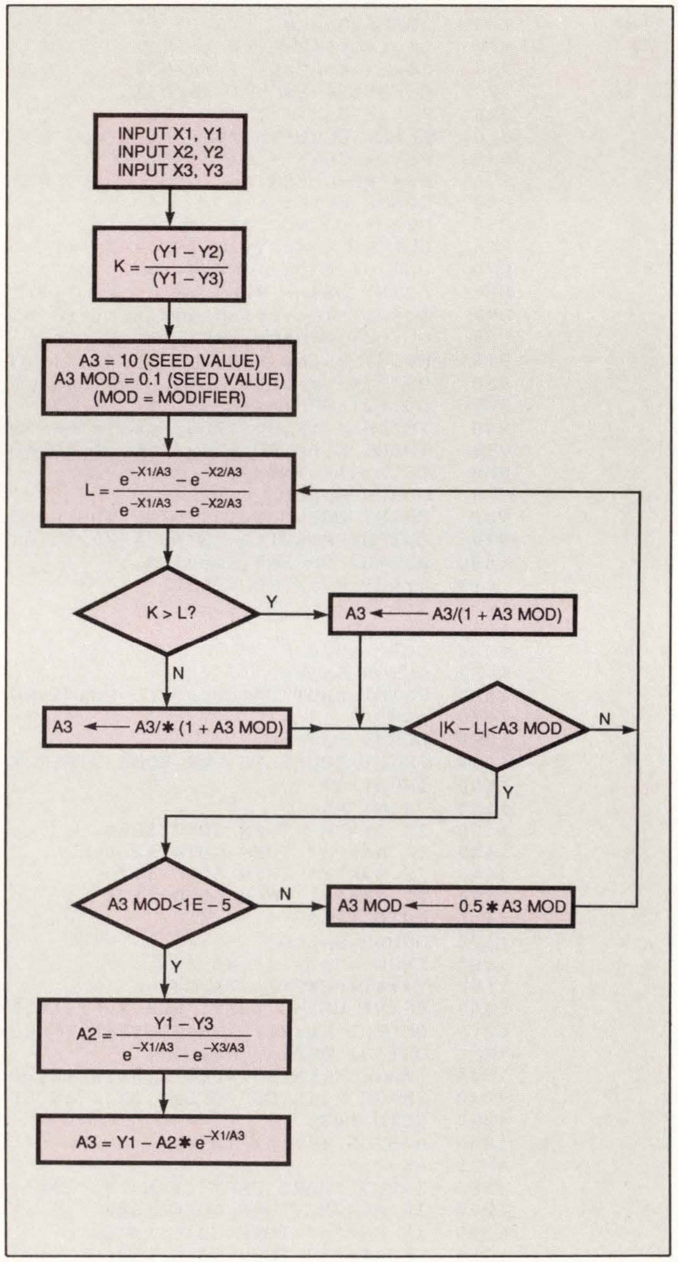


Fig 1—This flow chart diagrams the iterative calculations that fit an exponential function to real-world data.

Floppy-repair program saves files

K V Ramakrishnan
NPOL, Kerala, India

The program in **Listing 1**, written in TURBO C 2.0 (you can also obtain the **listing** from the EDN BBS (617-558-4241,300/1200/2400,8,N,1—from main menu, enter (s)ig, <s/di_sig>, rk978)), enables you to save files in the event that while examining the directory or accessing any file in your valuable floppy disk, MS-DOS displays the following, dreaded error message:

General Failure error reading device A Abort, Retry, Fail?

This error results from data corruption in sector one of track zero. This sector consists of format-dependent data like the DOS version, number of sectors, and sides used for formatting. If you can write this data back, you can re-use that floppy without the loss of files in the floppy. To perform the rewrite, you need a similar good floppy formatted under the same version and with the same switch settings. **Listing 1**'s program will ask you to place the good floppy in drive A. It will read the data in the first sector and place this data in the bad floppy when you place it back in drive A.

EDN BBS /DI_SIG #978

To Vote For This Design, Circle No. 748

Listing 1—Floppy repair program

```
 /* Floppy Repair Program written in TURBO C 2.0 */

int err=0xaa,head,track,sectors,nsector,test=0;
unsigned char buff[516],chr;
main()
{
printf("\n Insert good Floppy in drive A. OK ?(y/n)");
chr = getche();
if(((chr & 0xff)=='Y')||((chr & 0xff)=='y')){
   nsector = 1;head = 0;track = 0; sectors = 1;
   readwrite(0);    /* read disk */
   printf("\n Replace good floppy with bad one. OK ?(y/n)");
   chr = getche();
   if(((chr & 0xff)=='Y')||((chr & 0xff)=='y')){
     test = 0; readwrite(1); /* write disk */
     }
 }
}

readwrite(int rw)
{
if(rw == 0) /* read disk */
    err = biosdisk(2,0,head,track,sectors,nsector,&buff);
else         /* write     */
    err = biosdisk(3,0,head,track,sectors,nsector,&buff);
switch(err){
   case 0x00: printf("\n operation successful");              break;
   case 0x01: printf("\n Bad command          ");              break;
   case 0x02: printf("\n address mark not found");            break;
   case 0x04: printf("\n record not found    ");               break;
   case 0x05: printf("\n reset failed         ");              break;
   case 0x07: printf("\n drive parameter activity failed");   break;
   case 0x09: printf("\n attenpt to DMA across 64K boundary");break;
   case 0x0b: printf("\n bad track flag detected");           break;
   case 0x10: printf("\n bad ECC on disk read");              break;
   case 0x11: printf("\n ECC corrected data error");          break;
   case 0x20: printf("\n controller has failed");             break;
   case 0x40: printf("\n seek opereation failed");            break;
   case 0x80: printf("\n attachment failed to respond");      break;
   case 0xbb: printf("\n undefined error occurred");          break;
   case 0xff: printf("\n sense operation failed");            break;
   default  : if(test++ <5 ) readwrite(rw);
              else printf("\n operation failed");
   }
}
```

FEEDBACK AND AMPLIFICATION

Corrections

For "One coax cable carries video and power" by Jeff Kirsten and Charlie Allen (EDN, March 14, 1991, pg 137), Q_2 in **Fig 2** should be a 2N3904, not a 2N3906. In **Fig 3**, the ground symbol at IC_2 should be a triangle, for power-supply ground.

Author Jim Honea points out that in the write-up of his Design Idea, "MOV model spoofs Spice" (EDN Software Engineering Special Supplement, March 28, 1991, pg 35), K, not α, can be as low as 1×10^{-74}.

EDN's bulletin board is on line

Call EDN's free bulletin board system (BBS) at (617) 558-4241 (1200/2400,8,N,1) and select /DI_SIG to get additional information or to comment on these Design Ideas.

Software Development Tools For MC68EC μPs

- *C compilers provide ANSII conformance*
- *In-line assembly-language facilities for each μP*

Intertools software-development products are now available for Motorola's MC68EC000, MC68EC020, MC68EC040, and MC68330 processors. Release 8.0 of the tool kit includes in-line assembly, full ANSI conformance of the C compilers, and support for C++ preprocessor programs. Each tool is specifically tailored to take advantage of the hardware features of the target μP. The tools recognize the difference between capabilities of the MC68030 memory management unit and those of the MC68EC030 access-control unit; the MC68330 tools take advantage of the unique CPU32 instruction set and addressing modes. The tool kit includes Motorola-compatible macroassemblers that allow an unlimited number of relocatable, absolute, and combinable segments. The Compiler/Assembler/Utilities tool kit hosted on an IBM PC or compatible, from $1975. The XDB debugger is available for three execution environments: XDBice, integrated with popular in-circuit emulators, $1650; XDBrom, bundled with a configurable ROM monitor, $2500; and XDBsim, bundled with a 68000-based simulator board, $2400.

Intermetrics Microsystems Software Inc, 733 Concord Ave, Cambridge, MA 02138. Phone (800) 356-3594; in MA, (617) 661-0072. FAX (617) 828-2843. **Circle No. 351**

C Compiler And Simulators For Intel 8051

- *C compiler provides full ANSI conformance*
- *Simulator provides I/O simulation*

The Arch51 compiler kit for Intel's 8051 microcontroller consists of a C compiler kit, a simulator/debugger, and an I/O simulator. Archimedes Inc developed these tools, which are fully integrated with Intel's in-circuit emulator, the ICE-51FX/PC. This emulator plugs into a host PC computer and allows you to test any of more than 30 variations of the 8051 controller. The C cross-compiler, which runs on IBM PCs and compatibles, conforms to the ANSI standard; the compiler comes with a macro assembler, a linker/locator, a librarian, and C libraries. SimCASE, the simulator/debugger, provides both C and assembly source-level debugging facilities, as well as performance- analysis tools and a stimulus generator for on-chip I/O simulation. SimI/O is a programmable simulator that works in conjunction with SimCASE to perform modeling and analysis of external I/O. Arch51 Microcontroller C Kit, $1295; SimCASE, $995; SimI/O, $695.

Intel Corp, Box 58065, Santa Clara, CA 95052. Phone (800) 874-6835 or local office.

Circle No. 352

Enhanced Compilers

- *Code generators and optimizers boost system performance*
- *Debugging tools use Open Look GUI*

SPARCompilers for C, C++, Fortran, Pascal, and Modula-2 include improved code generators and optimizers. According to the vendor, benchmarks published by SPEC (Systems Performance Evaluation Cooperative) show that software compiled with the new versions shows a performance gain of as much as 18% over the same software compiled with previous versions. These versions come with dbx and dbxtool debuggers and Sun Sourcebrowser, and run under the NSE (network system environment) operating system, using the Open Look GUI (graphical user interface). NSE also provides version control, configuration management, and parallel development. A single license for the C, C++, Fortran, and Pascal compilers, $2000 each; for the Modula-2 compiler, $2200; a 20-user license for the NSE, $10,800.

Sun Microsystems Inc, 2550 Garcia Ave, Mountain View, CA 94043. Phone (415) 960-1300. FAX (415) 969-9131. **Circle No. 353**

OCR System For Windows 3.0

- *Provides SCSI interface for Micro Channel Architecture*
- *Converts data into multiple formats*

The Topscan Plus OCR (optical character recognition) system provides a SCSI interface for IBM PC/AT-compatible and IBM Micro Channel Architecture PCs and compatibles that run Windows 3.0. The system employs the vendor's CDP-6000 and CDP-9000 scanners, which have scan rates as high as 3.3 sec/page and character-recognition rates as high as 250 cps. The software automates as much as possible of each phase of the OCR process and includes a deferred processing feature that lets you split scanning and recognition into two separate tasks. A pop-up verifier helps to reduce proofing time. In addition, the software can convert the data to any one of more than 50 word-processing and spreadsheet formats. To run Topscan Plus, you need an IBM PC or compatible with 2M bytes of RAM, Windows 3.0, and a hard disk. Both scanners come equipped with high-volume sheet feeders and support RS-232C, SCSI, and Ethernet interfaces. Topscan Plus, with a CDP-6000 scanner, $17,450; with a CDP-9000 scanner, $29,450.

Calera Recognition Systems Inc, 2500 Augustine Dr, Santa Clara, CA 95054. Phone (408) 986-8006. **Circle No. 354**

Diagnostic Knowledge-Management Shell

- *Embeds diagnostic knowledge bases in existing software*
- *Lets you deliver a troubleshooting system for customer support*

Version 2.0 of Testbench is a diagnostic knowledge-management shell that allows you to integrate diagnostic applications with an existing software environment. Testbench consists of two components: Testbuilder, which is a graphical tool for capturing troubleshooting knowledge; and Testview, which is a delivery system based on a layered-software architecture. Each layer provides many options for integration and customization. New features of Testview include the ability to save and resume a diagnostic session at any point; the ability to query an external device and use the resulting data in the diagnostic session; and the ability to specify a repair and to resume diagnosis if the repair fails. Testbuilder runs on Sun workstations; $48,000 for the first copy. Testview runs on IBM PCs and compatibles, $1400.

Carnegie Group Inc, 5 PPG Pl, Pittsburgh, PA 15222. Phone (412) 642-6900. **Circle No. 355**

Network Control System

- *Default setups simplify installation*
- *Data conversion lets you use Sniffer trace files for analyses*

Protolyzer version 1.1 provides features that make it easier to take data off the network and present it in a useful format. The database, for example, lets you replace hexadecimal node addresses with the actual names that represent each workstation. For management purposes, you can also add office location, hardware in use, and other relevant information. The tools include standard applications for baselining, active monitoring, comparative troubleshooting, and load simulation of the network. You can analyze Token-Ring and Ethernet data and draw visual maps of the network. One tool converts Sniffer trace files into Protolyzer data files for extended analyses. From $6995.

Protools Inc, 14976 NW Greenbrier Pkwy, Beaverton, OR 97006. Phone (503) 645-5400. FAX (503) 645-3577. **Circle No. 356**

Network Virus Protection

- *Detects and eliminates more than 400 viruses*
- *Works with DOS, Windows, and networks*

Central Point Antivirus is a program that can recognize and eliminate more than 400 known and unknown viruses. The program provides file immunization that makes executable files self-protecting; it also includes a memory-resident module that provides the same level of protection as the main program.- When running under Windows 3.0, the program provides a dialog box to alert you to virus infections that may take place while Windows applications are running. The Vsafe memory-resident program monitors the system for illegal activities which could indicate that a virus is trying to infect a file. It creates and verifies checksums for each file to determine whether an unknown virus might be at work; a smaller program (Vwatch) checks only for known viruses. You can choose to run either Vwatch (8k bytes) or Vsafe (22k bytes), depending on how much memory is available. $129.

Central Point Software Inc, 15220 NW Greenbrier Pkwy, Suite 200, Beaverton, OR 97006. Phone (503) 690-8090. **Circle No. 357**

Segmented Linker For Generating .EXE Files

- *Runs under DOS, Windows, or OS/2*
- *Sets no limit on the number of items being linked*

The Optlink/Windows version 2.50 segmented linker can generate .EXE files for DOS, Windows, or OS/2 systems. It uses extended or expanded memory when available, and sets no limit on the number of modules, libraries, publics, or segments being linked. According to the vendor, the linker performs as much as 20 times faster than Microsoft's Link. Other features include the ability to copy .RES files to .EXE at link time; the ability to work with Codeview 3.0 and Windows 3.0; and compatibility with Microsoft segmented linkers for generating Windows and 16-bit OS/2 .EXE and .DLL files. In addition, the linker generates complete global cross-references and provides detailed segment mapping. New users, $350; upgrade for registered users, $75.

SLR Systems, 1622 N Main St, Butler, PA 16001. Phone (412) 282-0864. FAX (412) 282-7965.

Circle No. 358

Memory Manager

- *Allows access to DOS memory above 640k in enhanced 386 mode*
- *Needs no additional hardware to manage extended memory*

HI386 Complete allows you to access the DOS memory between 640k and 1M bytes while operating Windows 3.0 in 386 enhanced mode. Previously, you could access high memory only when Windows was running in real mode. The new version provides software-only memory management that eliminates any conflicts with the memory-management functions of Windows 3.0 because it makes use of the Microsoft virtual device driver (VxD). You can relocate network software, device drivers, DOS utilities, and TSR programs to high memory, thereby freeing as much as 224k bytes of main memory for memory-intensive applications. For registered users of HI386, upgrade to Windows-compatible HI386 at no charge; upgrade for users of any other memory manager, $29.95 plus a copy of the documentation cover; new users, $129.

RYBS Electronics, 2590 Central Ave, Boulder, CO 80301. Phone (303) 444-6073. FAX (303) 449-9259.

Circle No. 359

EDN PRODUCT MART

This advertising is for new and current products.

Please circle Reader Service number for additional information from manufacturers.

There is a Difference!

Lifetime Free Updates

CP-1128 $1295

A programmer is not just another programmer. That is why BP Microsystems is committed to bringing our customers the highest quality programmers at an afforable price.This commitment is evident in our CP-1128 Combination PROM/EPROM/PLD Programmer supporting over 1800 devices up to 28-pins. Call today!

BP MICROSYSTEMS

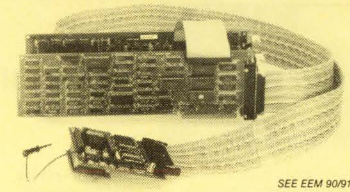
1-800-225-2102

713/461-9430

CIRCLE NO. 325

68HC11

PC-based emulator for 68HC11



SEE EEM 90/91 Pages D 1320-1323

- PC plug-in or RS-232 box.
- Pull-down menus with full window support, combined with command-driven User Interface.
- Up to 16 MHz real time emulation.
- No intrusions to the 68HC11's resources.
- 48 bit wide 16K deep trace. All functions usable without disturbing emulation. Time stamping. Two level trigger.
- Symbolic and C Source Level Debugging, including in-line assembler and disassembler.
- Supports A, E, D and F parts.

Prices: 64K Emulator and pod $2590;* 4K Trace $1995*

CALL OR WRITE FOR FREE DEMO DISK!

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

*US only

CIRCLE NO. 326

Logic Design Software

CUPL™

The Fastest Way Through Customs

Develop logic designs using the popular CUPL 4.0 PLD/FPGA development software with the newest and latest features that gets you through customs in a hurry! We can get you the product you need now, and in the future. CUPL 4.0 starts at $495.

For more information, call 1-800-331-7766. **LOGICAL DEVICES, INC.**

CIRCLE NO. 327

Professional Quality Tools for

80x86 ROM Development

Datalight gives you the tools you need for effective 80x86 ROM development:

ROM-DOS ROMable operating system. Compatible with MS-DOS 3.3. Operates with a disk system or a diskless environment. Less than 32K in size. Only $6 each in quantity!

C_thru_ROM full-featured ROM development kit. Includes Turbo and CodeView remote debugging, 80x86 locator, startup code, limited ROMable library and much more . . .

Complete information and *free demo disks.*

1-800-221-6630

Toll Free—Call Us Today!

Datalight

17455 68th Avenue NE, Suite 304, Bothell, WA 98011
(206) 486-8086 • fax (206) 486-0253

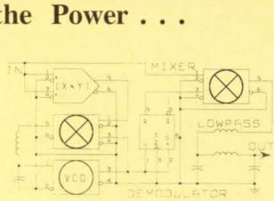
CIRCLE NO. 328

Exploit the Power . . .

Of. . .

PSpice®

Low Cost Mixed-Mode Simulation



NOW . . .

Links Schematic capture with Mixed-Mode simulation

. . . You have a choice!

Use low-cost PC's to multiply your productivity
Converts schematic input to simulation netlist
Simple schematic setups for worst case parametrics
Provides fast generation of icon driven subcircuits
Capabilities for linking to Concurrent Engineering
Simultaneous analog, digital and behavioral modeling
Project oriented directory & file management system
Design top-down or bottom-up - - ideal idea testing
So easy to use that it's actually fun to work

CALL FOR DEMOS AND DISCOUNTS
(503)645-8297

OrSPICE ™ CapSPICE ™ OOTMs ™

PSpice® is a trademark of MicroSim Corporation Irvine, California 92718

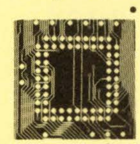
NW SILICON SPECIALISTS, INC 2700 NW 185TH AVE, STE 1200 PORTLAND, OREGON 97062

CIRCLE NO. 329

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 autorouting 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. 330

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

NEW! iceMASTER™
COP8 8051 68HC11

YOUR WINDOW TO EMULATION PRODUCTIVITY

- Easy to learn & use
- Windowed interface -- user configurable
- FAST! Download -- < 3 sec. typ. at 115KB
- Source Level debug

- **A 4K frame trace buffer** with advanced searching capabilities.
- **Hyperlinked On-line help** guides you through the emulation process.
- **iceMASTER** connects easily to your PC, requires no disassembly, or expansion slots. Works on any PC (DOS or OS/2), MicroChannel or EISA. Even laptops!
- **Supports more than 50 different 8051 family derivatives.** M68HC11 support will be available early in 1991.
- **Try iceMASTER risk free!** Satisfaction Guaranteed or return for a full refund!*
- **RENTALS AVAILABLE!** Ideal for consultants and researchers!
- **Call today for free demo disk and ask about a free 8051 Macro Assembler! (800) 638-2423**

MetaLink® Corporation

FREE ASSEMBLER

MetaLink Corporation P.O. Box 1329 Chandler, Az 85244-1329
Phone: (602) 926-0797 FAX: (602) 926-1198 TELEX: 4998050MTLNK
* With 10-day trial period

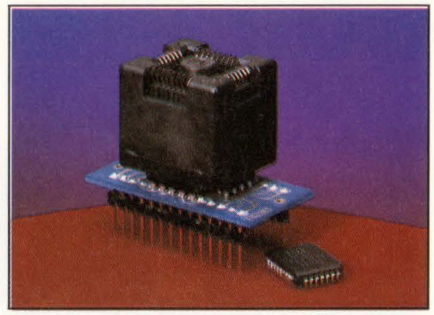
CIRCLE NO. 331

New SUPERPRO™
UNIVERSAL PROGRAMMER

- programs • PAL, EPLD, GAL, PEEL, FPL (up to 68 pin PLCC)
  - E(E) PROM, Flash EPROM up to 4Mbits (40 pins)
  - Microcontroller, Bipolar PROM.
- Tests TTL/CMOS Logic, D/S Memory Device.
- High speed parallel interface card to PC/XT/AT/386
- Pull-down Menu driven, Library Operating software.
- **Fast Device update** on user's request
- 40-pin **Gold ZIF** Socket
- Lifetime **Free Updates** (BBS)
- User Device Library Generator (optional)

TOLL FREE 1-800-541-1975
XELTEK
764 San Aleso Ave.
Sunnyvale, CA 94086
TEL (408) 745-7974
FAX (408) 745-1401

CIRCLE NO. 332



PAL/PROM Programmer Adapters

- Any EPROM programmer designed for DIPs can be converted to accept LCC, PLCC, and SOIC sockets in seconds!
- To program, just insert an Adapt-A-Socket™ between the programmer's DIP socket and the circuit to be programmed.
- Designed to fit all types of EPROM programmers, including Data I/O 120/121A, Stag, Logical Devices, etc.
- Quick turnaround on custom engineering services, if needed. For a free catalog, contact:

Emulation Technology, Inc.
2344 Walsh Ave. Santa Clara, CA 95051
Phone: 408-982-0660 FAX: 408-982-0664

ET

CIRCLE NO. 333

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. 334

ROM-IT
EPROM EMULATION SYSTEM

NEW 4-MEGABIT VERSION

- Emulates up to 8 4-Megabit EPROMS with one control card.
- Downloads 2-Megabit programs in less than 23 seconds.
- Allows you to examine and modify individual bytes or blocks.
- Accepts Intel Hex, Motorola S-Record and Binary files.
- Software available for IBM PC and compatibles and Macintosh systems.
- Base 27256 EPROM System **$395.00** Other configurations available.

ORDER TODAY--IT'S EASY
CALL OR FAX FOR MORE INFORMATION

Incredible Technologies, Inc.
(708) 437-2433
(708) 437-2473 Fax
VISA now accepted.

CIRCLE NO. 335

Now Networkable!

Facts about

500,000

ICs and Semiconductors at Your Fingertips

Cahners CAPS is the newest component search and selection tool for electronic design engineers:

- PC-driven, CD-ROM-based
- Includes unabridged manufacturers' datasheets
- Represents more than 450 manufacturers worldwide

Call toll-free: 1-800-245-6696

CAHNERS CAPS Computer Aided Product Selection

275 Washington Street
Newton, MA 02158-1630
Telephone: 617-558-4960
Facsimile: 617-630-2168
Telex: 940573

CIRCLE NO. 336

DIRECT CODING

PROGRAMMING WITHOUT LANGUAGE

Software made ***easy***.

Professionals, Scientists, Engineers, Technicians.

M-Code Personal $99.95
A tool to write software for the **8088** family, **8051** family and other processors
Call or send for data sheet

DOSystems, (408) 625 9016
Box 4601 Carmel CA 93921

CIRCLE NO. 337

NEW! iceMASTER™
COP8

FULL-FEATURED EMULATION SYSTEM FOR COP8

- PC-hosted COP8 emulator system.
- Easy to learn and use.

- **iceMASTER is fast.** 8K file loads in less than a second with 115.2K baud link using standard COMM port.
- **iceMASTER COP8** connects easily to any PC, requires NO disassembly or expansion slots. Works on PC (DOS or OS/2), Micro Channel, or EISA. Even Laptops!
- **iceMASTER is flexible.** Windowed interface - user configurable with pull-down menus, combined with hot-keys, context sensitive hyperlinked help, on-screen editing.
- **iceMASTER is powerful.** 4K frame trace buffer with advanced searching and filtering capabilities.
- **iceMASTER is versatile.** One iceMASTER COP8 allows emulation of more than 10 different COP8 family derivatives via interchangeable probe cards.
- **Call us today for a FREE demo disk.**

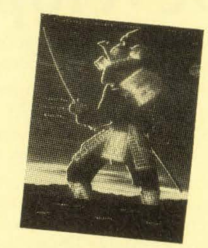
MetaLink® Corporation
(800) METAICE
(800) 638-2423

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

CIRCLE NO. 338

TOOLS FOR ELECTRONIC WARRIORS

- SCHEMATIC CAPTURE
- GRAPHICS
- PCB LAYOUT
- SIMULATION
- ROUTERS
- CAD/CAE & MORE!



For Your Free Catalog Call
1-800-743-7074

Tools For Technical Professionals
TechExpress
31200 LaBaya Drive • Suite 301 Westlake Village, CA 91362
EDM620

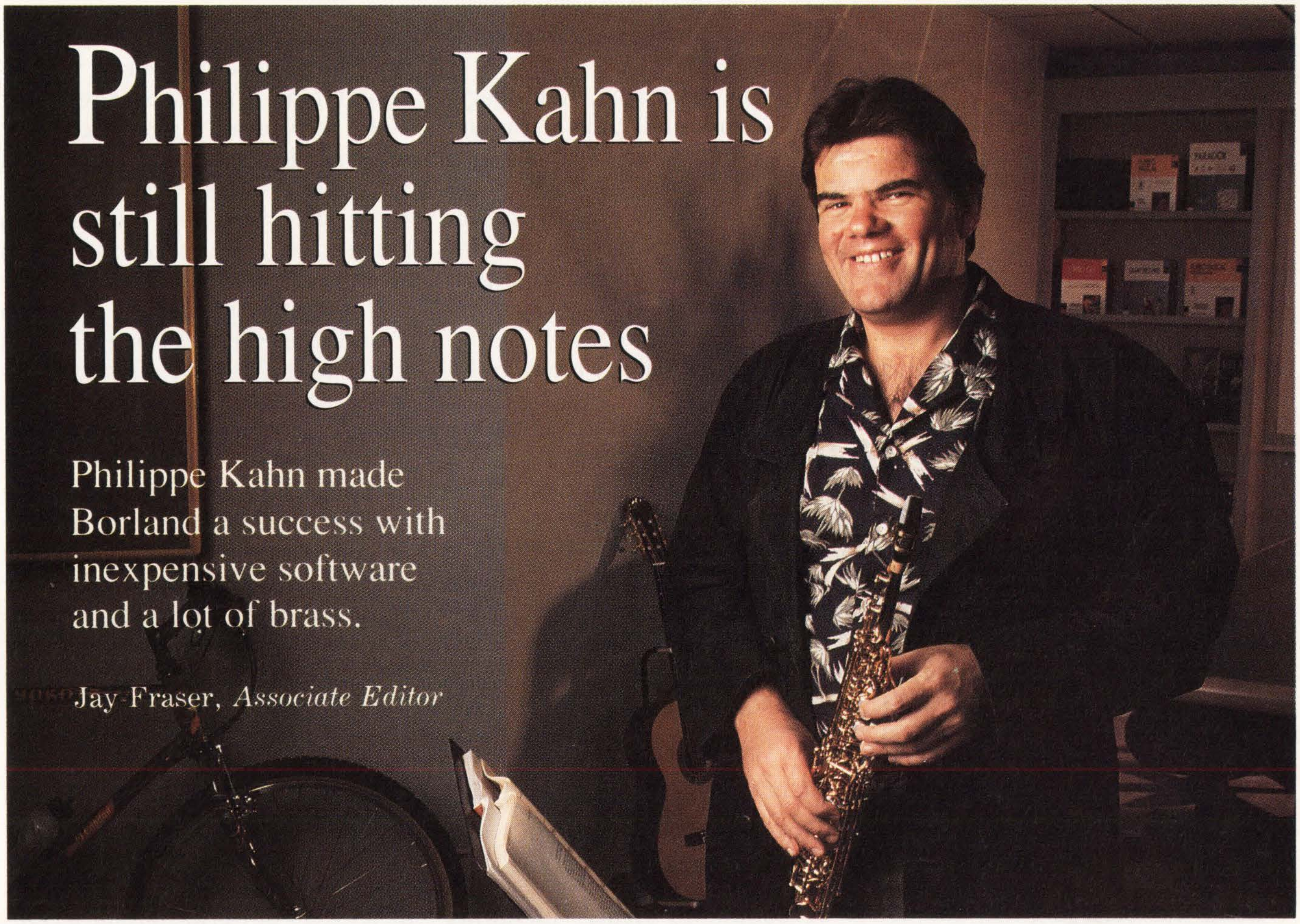
CIRCLE NO. 339

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

Philippe Kahn is still hitting the high notes

Philippe Kahn made Borland a success with inexpensive software and a lot of brass.

Jay Fraser, *Associate Editor*



Photography by Christopher Springmann

In August of 1983 Philippe Kahn was in a serious bind. Four months earlier he had borrowed some money to start his software company, Borland International, and rent a cramped, two-room office above a garage in Scotts Valley, CA. He was the only full-time person on the staff and he had only one product—a compiler he and some friends had developed and named Turbo Pascal. But Kahn had no distributors for Turbo Pascal and the borrowed money was rapidly running out. To add to his problems, after immigrating from France the year before he had been unable to obtain a work permit or a green card. He was an illegal alien.

Kahn came up with the idea of selling Turbo Pascal by direct mail, an unconventional approach at that time. He wrote an advertisement that he wanted to run in *Byte* magazine, but he didn't have enough money to pay for it. So he decided to stage an elaborate ruse.

He invited a salesman from *Byte* to come to his office. Then he prepared a chart that supposedly showed Borland's media plan for the coming year. Kahn listed every computer magazine he could think of and drew a large X through *Byte*. He put the chart on his desk where the salesman couldn't help but see it. Kahn also paid a couple of people to bustle around the office for an hour so that it would look busy and arranged for some friends to call him up so the phone would ring constantly.

When the salesman arrived Kahn apologized and said they had decided not to go with *Byte* after all. The salesman noticed the phony chart on Kahn's desk and said, "Wait a minute, why don't you want to advertise in my magazine?" Kahn replied that he didn't think it reached the right people. The salesman, afraid of losing out to his rivals, offered to make a deal. Kahn quickly got the terms he wanted. The ad ran on credit, payment due in 90 days.

"I didn't have very high expectations," says Kahn. "I hoped to pay for the ad and pay the rent. It was a real surprise when I received about $100,000 worth of orders in the first month."

Taking risks, trying new approaches, and displaying a lot of nerve have continued to characterize Kahn's style. The burly president, CEO, and chairman of Borland has gained notoriety because of his feuds with reporters, criticism of American management methods, and caustic comments about competitors. His sometimes abrasive manner has earned him some detractors, but his ebullience and joie de vivre have won him a large number of supporters. He's been called the flamboyant Frenchman, the Great Kahn, and the enfant terrible of the software industry.

Kahn has a knack for attracting attention. At one computer conference he walked into a swimming pool fully dressed. At another conference he threw a now legendary toga party, at which he serenaded his guests on his saxophone for hours. Kahn claims these things were calculated to draw attention to Borland and he only did them for the good of the company, but he isn't 100% convincing.

In the past, Kahn was just as likely to greet visitors in a Hawaiian shirt and running shorts as a three-piece suit. But despite his iconoclastic image, few people question his skill as a businessman. From that shoestring operation above the garage in 1983, he has guided Borland to become, by its own figures, the fourth largest independent software company in the US. Today Borland has nearly 1,000 employees, and last year it took in more than $225 million in sales.

Kahn was born in Paris, and attended an experimental United Nations school with an international student body. He soon added fluent English, Spanish, and German to his native French. Until he went to college, Kahn was torn between being either a mathematician or a jazz musician. He earned the equivalent of a BS in mathematics at the University of Zurich. While there, he studied under Niklaus Wirth, who created the Pascal programming language. He went on to the University of Nice, where he received his advanced degrees and helped pay for his education by playing the piano, flute, and saxophone in local nightclubs.

While he was in Nice he bought an Apple II. "I had been studying with computers and playing with computers, but this was my first *personal* computer," says Kahn. "I was fascinated by the fact that it was all in a little box. You can write your program, you can run it, you can make it all happen. You control your destiny."

After he graduated, Kahn began teaching math full time at the university, but soon became bored with it. He realized what he really wanted to do was work with computers. Taking control of his destiny, he gave up his teaching position and bought a one-way plane ticket to San Jose, CA.

Kahn says he didn't arrive in the US intending to found a high-tech company and become a millionaire. He was just looking for a job programming or working on advanced operating systems because that's what he enjoyed doing. He knocked on doors all over Silicon Valley and most of them remained firmly closed. He stayed alive by repairing computers, making printer cables, and doing some consulting.

"You can write your program, you can run it, you can make it all happen. You control your destiny."

A couple of firms, including Hewlett-Packard, showed some interest in hiring him, but when they found out he had no green card or work permit they hastily withdrew their job offers. "The door at Hewlett-Packard opened up, then slammed back right in my face," says Kahn. He adds that he finally came to the conclusion that if he were the president of a company no one would ask him if he had a green card.

Kahn founded Borland International in May of 1983. He made up the name and used the word "international" because he believed it would make the company sound like part of a multinational, and therefore more important.

One factor that made his first program, Turbo Pascal, an instant hit was its extremely low price. While other companies were selling similar but slower programs for $500 to $600, Kahn priced Turbo Pascal at $49.95. This aggressive pricing stirred up the first, but certainly not the last, controversy to surround him. Critics charged that Kahn was using a gimmick to get a toehold in the industry. They said he couldn't make any money at those prices and he'd soon have to raise them. Kahn proved them wrong.

Buoyed by the success of Turbo Pascal, the following June, Borland introduced Sidekick, a personal information manager, also priced at $49.95. It quickly became a best seller as well. To date, Turbo Pascal has sold more than 500,000 copies and Sidekick has sold more than one million. Also controversial is Kahn's policy of never copy protecting any of his software. He has said he feels this would be "a tax on honesty."

Borland is an absolutely no frills company. Kahn nurtures a corporate culture he likes to describe as "barbarian." He wants small groups of people to work together to solve problems with no middle managers to get in the way, and he has vowed never to create a bureaucracy. His ideas seem to work. Borland's revenue per employee is approximately $300,000—one of the highest in the industry.

Borland's money and efforts go into research and development, not advertising or marketing. Although a substantial part of its income is still derived from direct-mail sales, all its ads are written and produced in-house, and there are no elaborate marketing plans. "We've never used an advertising agency or anything like that," says Kahn. "We try to find innovative angles to things, but we don't plan out every single ad that we run."

Kahn has readily embraced the California good life. He lives with his wife and two daughters in a spacious house in the hills near Santa Cruz complete with hot tub, recording studio, and a driveway crowded with expensive cars. He avidly pursues sports such as tennis, golf, and skiing, and he still plays music as often as he can. He and a group of his employees formed the Borland Turbo Jazz Band and perform at company functions.

Another of Kahn's passions is sailing, and Borland's rapid growth in the 1980s is reflected in the boats he owned. After the success of Turbo Pascal, before he purchased a house or car, Kahn bought himself a 20-ft sailboat. The following year, when Sidekick's sales soared, he moved up to a 37 footer. In 1986 Borland went public and Kahn acquired a 43-ft boat. By 1988, the company was doing so well that Kahn treated himself to a sleek, 70-ft racer. That year he and his crew won the Pacific Cup Race from San Francisco to Hawaii in a record time of nine days and six hours.

During a trip back to Paris in 1985, Kahn took the time to stop by the American embassy and pick up the necessary papers to finally get his green card. More than two years after Borland was launched, he could finally work legally in the United States.

Over the years Kahn has been fearless in taking on the giants of the software industry. In 1987, Borland acquired Ansa Software Inc, which produced a database program called Paradox. Kahn saw that Paradox could compete directly with Ashton-Tate's dBase III. Following his usual strategy, he priced it at $149, about half of what dBase III costs. In just four years, Paradox became Borland's biggest seller and captured 20% of the database market.

Also in 1987, Borland introduced Quattro, a spreadsheet program designed as a direct challenger to Lotus 1-2-3. Kahn launched Quattro with a special price of $79.95, less than one third of 1-2-3's price. Two years later Borland came out with an upgrade called Quattro Pro and made a special offer—customers could buy the new program for $99.95 if they could show they owned the first version of Quattro or any version of 1-2-3. Borland now holds about 20% of the spreadsheet market.

Borland is already one of the major players in the software industry, and it's going to get larger very soon. The company has subsidiaries in Europe, Asia, and Australia, and Kahn recently signed an agreement to sell software in the Soviet Union. Borland has become the multinational corporation it pretended to be when it was founded.

The inroads Kahn has made into long-established markets have not gone unnoticed, and his marketing strategy for Quattro has gotten him embroiled in a controversy that continues to rage in the software industry—copyrighting user interfaces.

Last year Lotus won a lawsuit against Paperback Software International for copyright infringement. The judge ruled that the user interface of Paperback's VP-Planner imitated the user interface of 1-2-3 too closely. After that decision was handed down, Lotus decided to sue Borland.

"I think the software industry is at the turning point of a new era, and object-oriented technology will change its entire character."

Borland's Quattro has an alternative menu structure very similar to 1-2-3's. The program can run without it, but it enables 1-2-3 users to switch over to Quattro very easily. Lotus claims this violates their copyright. Kahn claims it doesn't.

"We believe we are perfectly within our rights for what we did, and that Lotus's lawsuit is without merit," says Kahn. "The US Copyright Office has stated for years that menu structures and commands are not copyrightable. If you are a software developer, what else are you to do but follow the US Copyright Office's guidelines?"

Kahn almost single-handedly created the market for low-cost, utility software, and he sees profound changes ahead. "The equivalent to the microprocessor revolution in the hardware industry is object-oriented technology in the software industry," he says. "I think the software industry is at the turning point of a new era, and object-oriented technology will change its entire character. It will help solve the problems of the glut of software and the bottleneck of developing software. As a consequence it will probably help the hardware industry too."

Borland and Kahn have both grown and matured over the last decade. These days Kahn is seen more often in a three-piece suit than a Hawaiian shirt. But he still can't resist getting in a good shot every once in a while.

During the last presidential election campaign in France, one of the candidates, Jacques Chirac, asked Kahn to appear with him on French television via satellite to discuss entrepreneurship. Kahn agreed. Then Chirac made the mistake of sending Kahn a script for the show. Kahn bought a red, white, and blue parrot—the colors of the French flag—and sent it to Chirac with a note that said, "Here's the perfect guest for your TV program." (Kahn eventually appeared on the show, but without a script.)

Today Kahn's office in the roomy new Borland headquarters is larger than the original two-room office over the garage. Along with computers, software, and books on programming, his office contains sports equipment and musical instruments. In less than 10 years Kahn has gone from being an illegal alien, scrambling to find work, to being the head of a powerful multinational corporation. Kahn may be a little mellower, but he hasn't really changed. He's still likely to pick up his one of his instruments and head off. "Sometimes I'll go to a place down the road and jam, just to break the routine." EDN

Article Interest Quotient
(Circle One)
High 515 Medium 516 Low 517

CAREER OPPORTUNITIES

1991 Recruitment Editorial Calendar

Issue	Issue Date	Ad Deadline	Editorial Emphasis
Magazine Edition	July 18	June 26	Product Showcase—Volume II • Test & Measurement, Computer Peripherals • Components, CAE/ASICs •
News Edition	July 25	July 5	ICs & Semiconductors, Peripherals**, Regional Profile: Massachusetts**
Magazine Edition	Aug. 5	July 11	CAE • ASICs, Test & Measurement • Computers & Peripherals • Technical Article Database
News Edition	Aug. 8	July 19	CAE, Datacom**
Magazine Edition	Aug. 19	July 25	Military Electronics Special Issue, Image Processing • Ultra High Speed ICs/ASICs • Computer Peripherals, Software •
News Edition	Aug. 22	Aug. 2	Peripherals/Components, Test & Measurement**, Regional Profile: Idaho, Colorado, Utah**
Magazine Edition	Sept. 2	Aug. 8	ASICs Special Issue, Semicustom ICs • CAE, Packaging • ICs & Semiconductors Data Converters
News Edition	Sept. 5	Aug. 16	Military Electronics Special Issue, Computer Architectures, Defense Electronics**
Magazine Edition	Sept. 16	Aug. 21	DSP/Microprocessors, ICs & Semiconductors, CAE/ASICs, Environmental Engineering • Software
News Edition	Sept. 19	Aug. 29	RISC/ICs, Computers**, Regional Profile: Florida**
Magazine Edition	Oct. 1	Sept. 5	Computers & Peripherals/Networks, DSP Chip Directory • ICs & Semiconductors/Memory Technology, Instrumentation
News Edition	Oct. 3	Sept. 13	ICs & Semiconductors, Multimedia**
Magazine Edition	Oct. 10	Sept. 19	Test & Measurement Special Issue, Oscilloscopes, VXI Board Directory • CAE/ASICs, Sensors & Transducers •
News Edition	Oct. 17	Sept. 27	ATE/Board & IC Testing, Artificial Intelligence**, Regional Profile: New Mexico & Arizona**
Magazine Edition	Oct. 24	Oct. 3	Telecommunications ICs, Graphics & Video Circuits, Computers & Peripherals, Software, Wescon Preview Issue
Magazine Edition	Nov. 7	Oct. 17	High Performance DSPs • CAE/ASICs, Computers & Peripherals/Communications, Software, Wescon Show Issue
News Edition	Nov. 14	Oct. 25	Telecommunications**, Wescon Show Issue
Magazine Edition	Nov. 21	Oct. 31	18th Annual Microprocessor Directory • Test & Measurement, CAE/ASICs, ICs & Semiconductors

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

HONEYWELL: OPENING THE DOOR TO AVIONICS TECHNOLOGY OF THE 90s.

Honeywell in Phoenix offers a variety of career opportunities in our Commercial Flight Systems Group. Our continuing growth has created the following positions:

Systems Design Engineer — In this area, you will be involved in guidance and control systems analysis and hardware/software design trade-offs. Specification designs, including guidance, navigation and control algorithm development, as well as systems integration and installation, flight test and customer liaison activity, are a part of these positions.

System Software Development — This area involves development of flight software for advanced guidance and control systems for aircraft using modular and structured programming techniques. You will be involved with algorithms and development of real-time programs in both assembly (8086 family Z8002, 68000) and high order languages such as Pascal, "C," Ada and PLM/86, with subsequent hardware integration.

Electronics Engineering — These positions involve the development of new processor/bus architectures and specifications to support fault tolerant/redundant airborne applications.

Display Systems — These positions offer systems, software and hardware opportunities with CRT/LCD display technology. You should be familiar with digital hardware design and/or real-time programming. Systems functions include overall system definition, design and customer interaction.

To qualify for the positions listed above, you should have a BSEE or a BSCS degree and at least three years of experience.

Quality Engineering — To qualify for this position, you should have a BS degree in an engineering curriculum. A minimum of two to five years of experience in quality engineering/assurance, reliability and/or product engineering is required. You should have computer applications experience. Customer interface experience is preferred.

Additional opportunities are available in:

- CRT/LCD Display Technology
- Avionics Systems Simulation
- CAE Engineering (Apollo Mentor Systems)
- Artificial Intelligence
- VAX Systems Administration
- Fiber Optic Pressure Sensors
- EMI/HERF
- Software Tools Development

Make a career move. Honeywell offers you a competitive salary and benefits package. All new employees are required to successfully complete a drug screening test. Send your resume and salary history, in confidence, to Honeywell, Commercial Flight Systems Group, Professional Employment (EDN-E845), P.O. Box 21111, M/S I-17C, Phoenix, AZ 85036.

Honeywell

HELPING YOU CONTROL YOUR WORLD

Equal Employment Opportunity/Affirmative Action Employer
U.S. citizenship required for some positions

ENGINEERS

PEOPLE DEDICATED TO EXCELLENCE

Tellabs, Inc., is a leading designer, manufacturer, and marketer of voice and data communication networking equipment used by telephone companies, long-distance carriers, and businesses in public and private communication networks throughout the world. Since we opened our doors 17 years ago in Lisle, IL, in Chicago's West Suburban Research and Development Corridor, we have developed a reputation as a company committed to high-quality products and service. We are currently looking for self-motivated, enthusiastic people who enjoy the challenges offered by the informal, fast-paced environment of an expanding company.

SOFTWARE ENGINEERS (MTS, SMTS, MGR)

BS or MS in CS/EE for development of high speed digital cross connect product lines. Positions require a minimum of 3 years of software development in a Unix/C environment and experience in one of the following areas: real-time operating systems, relational data base management, testing, 68xxx firmware design, software quality assurance, or software metrics. Knowledge of SONET or ISDN applications desired.

SOFTWARE ENGINEERS (MTS/SMTS)

BSCS for design and development of an ORACLE based network management system. Experience with ORACLE, Unix and C are essential.

HARDWARE ENGINEER (MTS)

BSEE to perform electronic circuit design for statistical and time division multiplexer. Requires 3-4 years of electronic circuit design and experience with 68xxx processors.

APPLICATIONS ENGINEER

Individual will design, manage the installation and support of telecommunication networks. Strongly prefer experience in generating detailed specifications for Bell Operating Company Central Office Equipment.

PRODUCT SPECIALIST

Individual will serve as first line support for OEM contracts and second line support for Field, International Marketing, and Distributors. Position requires three to five years experience with network related products. Prefer knowledge of CEPT and SONET.

DEVELOPMENT MANAGER (SAT)

Planning and coordinating the implementation of System and Applications Testing for Telecommunications Equipment. Requires six to ten years experience in product test development or product design coupled with two to four years supervisory experience.

VLSI DESIGN

Develop specifications, design, simulation and testing of application specific IC's. Requires BSEE/MSEE with concentrations in digital circuit IC design.

We offer opportunities for advancement, a comprehensive salary and benefits package. Send/Fax your resume and salary requirements (principals only) in confidence to: Human Resources, Department J-26, Tellabs, Inc., 4951 Indiana Avenue, Lisle, IL 60532.

Fax #708-969-7314

tellabs® *YOUR NETWORKING PARTNER.*™

An Equal Opportunity Employer

Unix is a Trademark of AT & T

Professional Profile

Announcing a new placement service for professional engineers!

To help you advance your career, Placement Services, Ltd. has formed the EDN Career News Databank. What is the Databank? It is a computerized system of matching qualified candidates with positions that meet the applicant's professional needs and desires. What are the advantages of this new service?

- It's absolutely free. There are no fees or charges.
- The computer never forgets. When your type of job comes up, it remembers you're qualified.
- Service is nationwide. You'll be considered for openings across the U.S. by PSL and its affiliated offices.
- Your identity is protected. Your resume is carefully screened to be sure it will not be sent to your company or parent organization.
- Your background and career objectives will periodically be reviewed with you by a PSL professional placement person.

We hope you're happy in your current position. At the same time, chances are there is an ideal job you'd prefer if you knew about it. That's why it makes sense for you to register with the EDN Career News Databank. To do so, just mail the completed form below, along with a copy of your resume, to: Placement Services, Ltd., Inc.

IDENTITY

Name ______

Home Address ______

City ______ State ______ Zip ______

Home Phone (include area code) ______

PRESENT OR MOST RECENT EMPLOYER

Parent Company ______

Your division or subsidiary ______

Location (City, State) ______

Business Phone if O.K. to use ______

EDUCATION

Degrees (List)	Major Field	GPA	Year Degree Earned	College or University

POSITION DESIRED ______

EXPERIENCE

Present or Most Recent Position From To Title

Duties and Accomplishments Industry of Current Employer

Reason for Change

PREVIOUS POSITION

Job Title ______

Employer ______ From ______ To ______ City ______

State ______ Division ______ Type of Industry ______

Salary ______ Duties and Accomplishemnts: ______

COMPENSATION/PERSONAL INFORMATION * (optional)

Years Experience | Base Salary | Commission

Bonus | Total Compensation | Asking Compensation

Min. Compensation | Date Available

☐ I own my home. How long? ______ I rent my home/apt. ☐

☐ Employed ☐ Self-Employed ☐ Unemployed

☐ Married ☐ Single | Height ______ Weight ______

Level of Security Clearance | I Will Travel

☐ U.S. Citizen ☐ Non-U.S. Citizen | ☐ Light ☐ Moderate ☐ Heavy

☐ WILL RELOCATE ☐ WILL NOT RELOCATE ☐ OTHER ______

My identity may be released to: ☐ Any employer ☐ All but present employer

EDN Databank A DIVISION OF PLACEMENT SERVICES LTD., INC.

265 S. Main Street, Akron, OH 44308 216/762-0279

SOFTWARE PROJECT LEADER

Our client, a manufacturer of high-tech electromechanical systems, is in need of an experienced Software Professional to assume the lead in their newest product offering. The individual we seek must be capable of continued technical contribution and have the ability to orchestrate an entire project through to completion.

Familiarity with the following technology is a must:

- ☐ **386-486 HARDWARE**
- ☐ **OS-2 (VER 1.0 + 2.0)**
- ☐ **META-WINDOWS GRAPHICS**
- ☐ **DATABASE APPLICATIONS DEVELOPMENT**
- ☐ **EMBEDDED SOFTWARE**
- ☐ **REAL TIME; MULTI-TASKING SYSTEMS**

Additional responsibilities will include developing software for the control of robotic mechanisms, the interpretation of analog/digital measuring systems, algorithms, and database applications.

Our client offers excellent salary and comprehensive benefits. For consideration, send your resume in confidence to: **Southern Technical Search, 1620 South Federal Highway #850, Pompano Beach, FL 33062.**

Southern Technical Search, Inc.

DIRECTOR

OF RESEARCH AND DEVELOPMENT

This highly successful producer of electronic medical devices has an immediate need for a dedicated, dynamic individual to take on the management of our cutting-edge R&D staff in New Jersey.

To qualify for this key assignment, you must have a minimum of 7-10 years' related management experience, with emphasis on taking new products from concept to completion. A BSEE is essential (MSEE preferred). Experience with electronic instrumentation equipment is also required. Health care related design equipment background a plus.

In return, we offer a very competitive salary, truly superb benefits, and an unusually supportive environment. For prompt, confidential consideration, please send resume with salary requirements, to:

Box NT 644
10 West 20th Street, New York, New York 10011
Equal Opportunity Employer

IOMEGA™
Makers of the Bernoulli Box

LEADING THE WAY IN ADVANCED TECHNOLOGIES

Are you looking for a company that continually moves forward with exciting and dynamic technologies? A company that will appreciate your abilities and expertise? Then take a look at IOMEGA CORPORATION. We are known as a leading developer of high-tech removable mass storage devices and innovator in the field. But we also offer a prime Utah location with its dry, mild climate and an abundance of recreational activities amidst some truly beautiful scenery. Add to that affordable housing, excellent schools and big-city conveniences combined with small-town friendliness, and you have a combination that can't be beat. We currently have the following positions available for qualified candidates:

SOFTWARE ENGINEERS

- Experience with MAC OS internals
- Experience with Motorola 68000 assembly language
- "C" programming language
- Device driver experience
- BSCS
- 3+ years experience

DRIVE FIRMWARE ENGINEERS

- BSEE/BSCS/BSCE (Computer Engineering)
- MSCS preferred
- Assembly, "C" languages
- 3+ years experience
- Experience in disk or tape drives
- Controller firmware
- SCSI experience preferred
- Knowledge of DOS internals

ADDITIONAL OPENINGS

We have continuous openings for qualified experienced Engineers in the field of magnetic recording for data storage.

Along with our advanced opportunities and prime location, we offer an excellent salary and benefits package. Please send resume to: **IOMEGA Corporation, Professional Staffing, Code 0065-1042, 1821 West 4000 South, Roy, UT 84067.**

Qualified female and minority candidates are specifically encouraged to apply.
Equal Opportunity Employer M/F/H/V

ADVERTISERS INDEX

*Advertiser in US edition

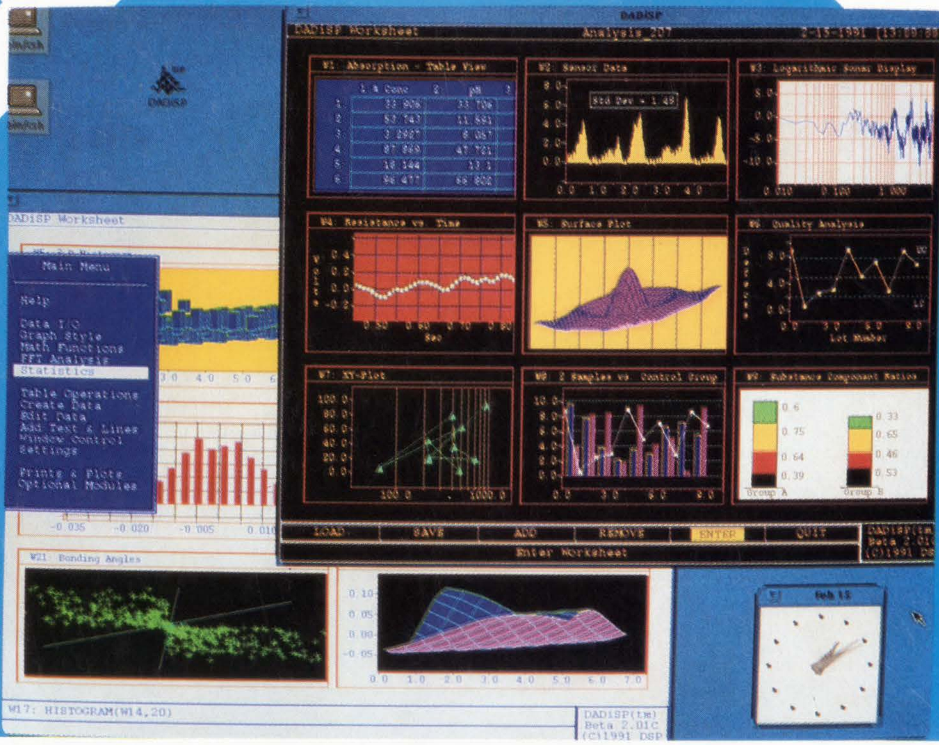
**Advertiser in International edition

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

Now under X-Windows

DADiSP 2.0

Now with Data-Acquisition Support



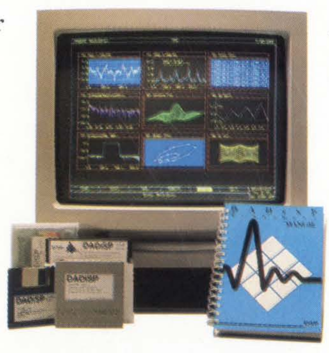
DADiSP.
The Engineering Spreadsheet

DADiSP — interactive graphics and data analysis software for scientists and engineers. DADiSP 2.0 delivers unprecedented power, through easy-to-use menus. Choose from hundreds of analysis functions and graphic views — from tables to 3-D. Simultaneously display multiple windows, each with different data or analyses, for unlimited perspective on your toughest data analysis problems.

Build your own analysis worksheets — build and display an entire signal processing chain, *without programming*. And DADiSP's powerful graphic spreadsheet automatically recalculates and updates the entire chain if you change your data or a processing step.

Do serious signal processing...the way you always pictured it! FFTs, digital filter design, convolutions, waterfall plots, and more — all at the press of a key.

Data Acquisition and Instrumentation Control — use DADiSP/LT and DADiSP/488 to collect data from A-D boards and instruments directly into a DADiSP window for immediate viewing and analysis.



Flexible, expandable, customizable — annotate your graphs and send them to printers, plotters, or publishing packages. Create your own macros, automate routine tasks, and run any program written in any language from within DADiSP. *DADiSP even lets you build your own menus.*

A proven standard — already used by thousands of engineers and scientists worldwide, in a whole range of applications like medical research, signal processing, chemistry, vibration analysis, communications, manufacturing quality control, test & measurement, and more. DADiSP supports the IBM PC and PS/2, SUN, DEC VAX, HP 9000, IBM RS/6000 and Concurrent families of personal computers and workstations.

GET THE PICTURE! 800-424-3131
IN MA 617-577-1133

Ask for our Evaluation Disk. For more information, write to DSP Development Corporation, One Kendall Square, Cambridge, MA 02139, or FAX: 617-577-8211.

DSP
Development Corporation

Australia-Interworld Electronics, 03-521-2952; Denmark-Dan Trade, 42-27-4511; Engberg Power Consult, 42-25-1777; England-Adept Scientific, (0462) 480055; Finland-Mespek OY, 9-0-3511800; France-SM2i, 1-34-89-78-78; Germany-Datalog, (02166) 46082; Digi-Logger, (07720) 38022; Infomatik Systeme, 0631-46007; Ingenieurburo Kohler- 069-769829; Kontron Messtechnik, 08165-770; India-Dynalog Microsystems, 22-583908; Italy-BPS Computers, 2-61290221; BRM Italiana, 11-771-0010; Japan-Astrodesign, 044-751-1011; Netherlands-Computer Engineering Roosendaal, 01650-57417; New Zealand-GTS Engineering, 09-392-464; Scotland-Diagnostic Instruments, 0501-43031; Sweden-Systec, 013-110140; Switzerland-Urech & Harr AG, 061-611325; Taiwan-Advantech, 2-351-2117; Howching (02) 5050525

CIRCLE NO. 25