june 2000 www.embedded.com volume 13, number 6

Embedded Systems® PROGRAMMING

Real Time: Analyzing Worst-Case Response Time

SOC Verification

Portable **Vector** Math Libraries

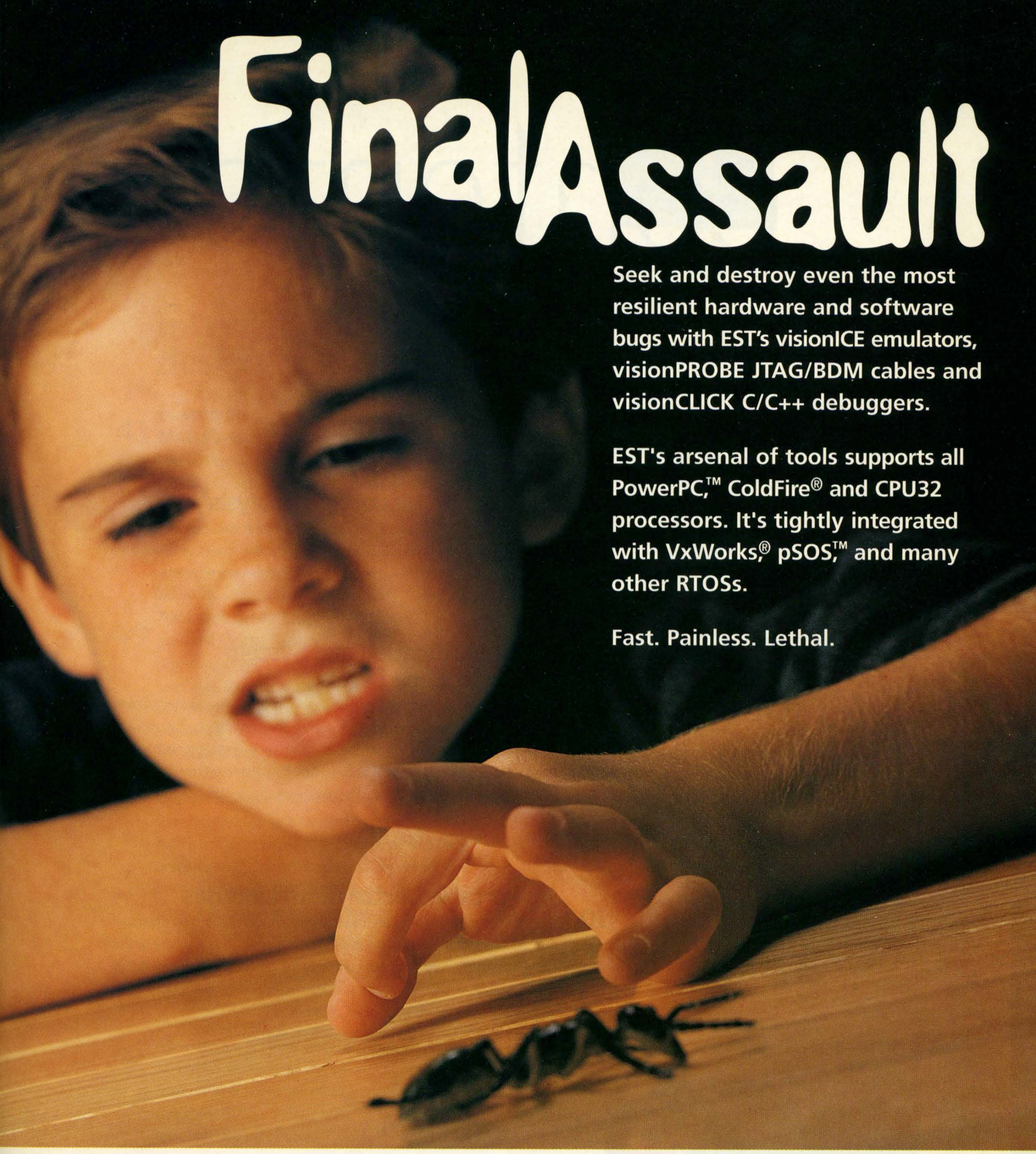
DSP and **Motion Control**

Internet Appliance Design:

Linux: Ready or Not

IrDA Protocol Layers

Compiled Java

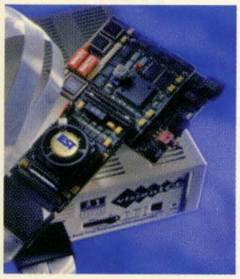


Final Assault

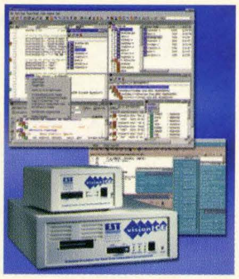
Seek and destroy even the most resilient hardware and software bugs with EST's visionICE emulators, visionPROBE JTAG/BDM cables and visionCLICK C/C++ debuggers.

EST's arsenal of tools supports all PowerPC,™ ColdFire® and CPU32 processors. It's tightly integrated with VxWorks,® pSOS,™ and many other RTOSs.

Fast. Painless. Lethal.



visionICE



visionCLICK

visionPROBE

PowerPC™

EST

Embedded Support Tools Corporation

Call 800-957-5588 or Download a Free Demo: www.estc.com

Embedded Systems
PROGRAMMING

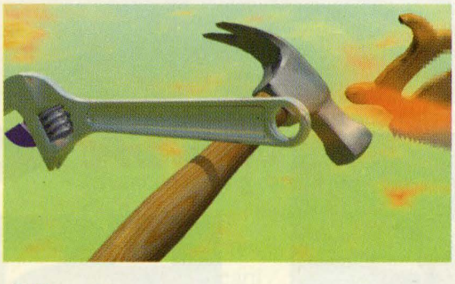
contents

JUNE 2000

COVER

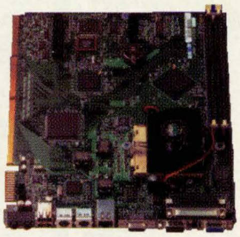
"Real time" may not always mean "real fast," but a late result is still a wrong result.

Cover illustration by Rupert Adley.



90

System abstraction levels are used by hardware and software designers for co-verification.



131

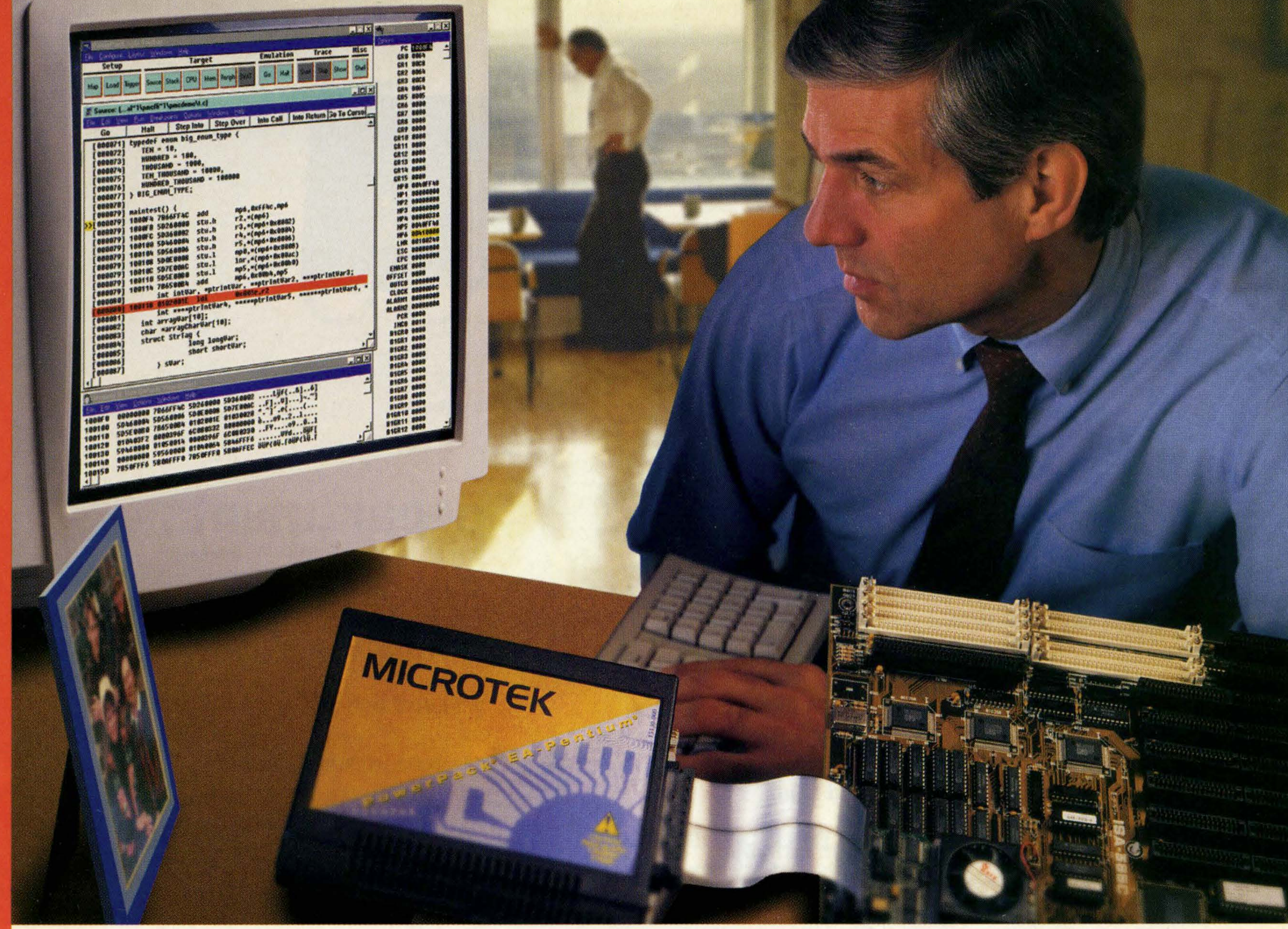
The SF810 micro NLX motherboard from RadiSys.

internet appliance design

departments

columns

Pentium® III • Celeron® • Pentium® • Pentium® II • Pentium® • Intel486™ • Intel386™EX/CX/SX • 683xx/HC16 • 8051/52/251



"If only I had spent less time on debug..."

I remember when I budgeted for my project. I refused to spend money on debug tools. I didn't think I would need them. The money I saved would make me a hero.

Now, my project is one month, two months ... six months overdue. Nobody thinks I'm a hero. What would I give to get some of that time back?

Microtek debug tools can make time. They speed up development and testing, saving you precious time at the end of the project, where you need it most!

Microtek emulators offer the debug features you need to track down and correct software, hardware, and system integration issues. They can find the errors software debuggers cannot see.

First, emulators can debug before the operating system is functioning. If an unexpected issue is affecting boot up, an emulator can find it. They are also operational after a hard crash. This is significant, because software debuggers lose debug information. Microtek emulators keep track of the last 128 KB of bus cycles, allowing you to sift through and find the problem.

Finally, if there is a subtle issue during the integration of hardware, Microtek EA emulators are capable of providing a clock-cycle-by-clock-cycle trace that allows you to view each signal and determine whether it was in the correct state. Microtek's tools are clearly superior to software debug solutions.

There has never been a project that couldn't use more time when it's critical ... like in final testing when everything has come together ... or in final debug, when everything is going down in flames.

Wouldn't it be great to deliver your next project on time? Next time, be sure to put a Microtek emulator in your project plan from the start.

It will help take the knot out of your stomach. And it will improve your company's bottom line!

URGENT!

If you are developing a Pentium class target using a reference design or off-the-shelf board, please take a moment to speak with our staff. They will insure your design is on track, and tool friendly if debug is needed in the future.

Additional interfaces: CAD/UL® and Windriver Tornado II®

MICROTEK
IN-CIRCUIT EMULATORS

1 (800) 886-7333
Phone (503) 533-4463
Fax (503) 533-0956
Email – info@microtekintl.com

www.microtekintl.com

ntium, Intel486 and Intel386 are registered trademarks of Intel Corp. © 1998 Microtek International

Financing Available

EmbeddedSystems
PROGRAMMING

EDITORIAL DIRECTOR
Lindsey Vereen, lvereen@cmp.com

MANAGING EDITOR
Felisa Yang, fyang@cmp.com

TECHNICAL EDITOR
Michael Barr, mbarr@cmp.com

SENIOR SPECIAL PROJECTS EDITOR
Tarita Whittingham, twhittingham@cmp.com

EDITORIAL ASSISTANT
Timothy Sullivan, tsullivan@cmp.com

CONSULTING TECHNICAL EDITORS
Jack G. Ganssle
Jerome L. Krasner, PhD

CONTRIBUTING EDITORS
Jack W. Crenshaw
Larry Mittag
Don Morgan
Dan Saks

VICE PRESIDENT/ELECTRONICS
Donna J. Esposito

EMBEDDED/DSP GROUP DIRECTOR
Mike Flynn, (415) 278-5251

PUBLISHER
Eric Berg, (415) 278-5220

EASTERN REGIONAL SALES MANAGER
Damon Graff, (781) 839-1285

EASTERN SALES REPRESENTATIVE
Jared Grimm, (781) 839-1286

CALIFORNIA SALES MANAGER
Andres Diaz, (415) 278-5274

CALIFORNIA SALES ASSISTANT
Molly Bruns, (415) 278-5298

WESTERN ACCOUNT EXECUTIVE
Sam Louis, (415) 278-5223

PRODUCTION COORDINATOR
James Whitehead

CIRCULATION MANAGER
Jennifer Schuler

CIRCULATION DIRECTOR
Susan Harper

SUBSCRIPTION CUSTOMER SERVICE
Toll free: (877) 676-9745
(847) 291-5215
esp@omeda.com
Back issues may be purchased on a prepaid basis through: Miller Freeman, 1601 West 23rd St., Suite 200, Lawrence, KS 66046; (800) 444-4881; (785) 841-1631

REPRINTS
Stella Valdez, (916) 983-6971

ONLINE PRODUCTION COORDINATOR
Billy Biondi, wbiondi@cmp.com

PRESIDENT/CEO, CMP MEDIA INC.
Gary Marshall

EXECUTIVE VICE PRESIDENTS
Regina Starr Ridley John Russell
Steve Weitzner Tony Uphoff

SENIOR VICE PRESIDENT/
GLOBAL SALES & MARKETING
Bill Howard

SENIOR VICE PRESIDENT/
BUSINESS DEVELOPMENT
Pam Watkins

PRESIDENT/ELECTRONICS
Steve Weitzner

Consumer Audit BPA

CMP

Visit our Web site at
www.embedded.com

Lindsey Vereen

Commie Penguins?

"They must be a bunch of communists," my wife Jan said one morning over breakfast as I vainly tried to explain the open source movement and the people who were embracing it as a business model. "How much capital does it take to start a company if you're going to give away your product?" she asked, not unreasonably. So I went to Jim Ready, the developer of the VRTX RTOS and now the president of MontaVista Software, which claims to be "dedicated to delivering open source software solutions for the worldwide embedded software market."

He pointed out that open source means *available*, not *free.*

The open source movement owes much to Unix. Born in Bell Labs, Unix became available to universities during the 1970s, and graduating computer science majors went forth into the world spreading the Unix gospel. Early Unix ran on DEC computers—principally PDP-11s—but versions were soon ported to the Motorola 68000 family, the Zilog Z8000, and the Intel x386. Unique among operating systems of the time, Unix was written in C and was portable to any target for which there was a C compiler. Moreover, it benefited from community development and grassroots distribution.

With everyone hacking it, Unix was not a static entity. Several variants of Unix popped up, ranging from AIX (from IBM) to Xenix (from Microsoft). Squabbles arose over what constituted the One True Unix. Initially, the battles were between proponents of the Berkeley Software Distribution (BSD) and AT&T's System 5, from which BSD Unix had begun to diverge. They converged again in System 5, Release 4 (SVR4). When that happened, commercial interests lined up against each other (AT&T and Sun on one side, and Apollo, DEC, HP, and IBM on the other) to hinder a single Unix from prevailing. Politicization eventually inspired Richard Stallman at MIT to found the Free Software Foundation, which led to the GNU tools and eventually to Linux.

In the embedded space, as elsewhere, Linux and open source are attractive to developers, and perhaps for that reason are rapidly turning into marketing concepts, about which everyone seems to be taking a position. Some vendors are generating fear, uncertainty, and doubt by saying mixing proprietary and open code could force you to have to offer your proprietary code to everyone. Other vendors are playing the wait-and-see game, and still others are already making engineering commitments.

Even if suppliers embrace Linux, they all have their eye clearly on their own intellectual property. That (along with service) is where their revenue is likely to come from, not from freely distributed source code. More than one effort is underway to add real-time capabilities to Linux. Other efforts are aimed at pruning it to fit in an embedded system.

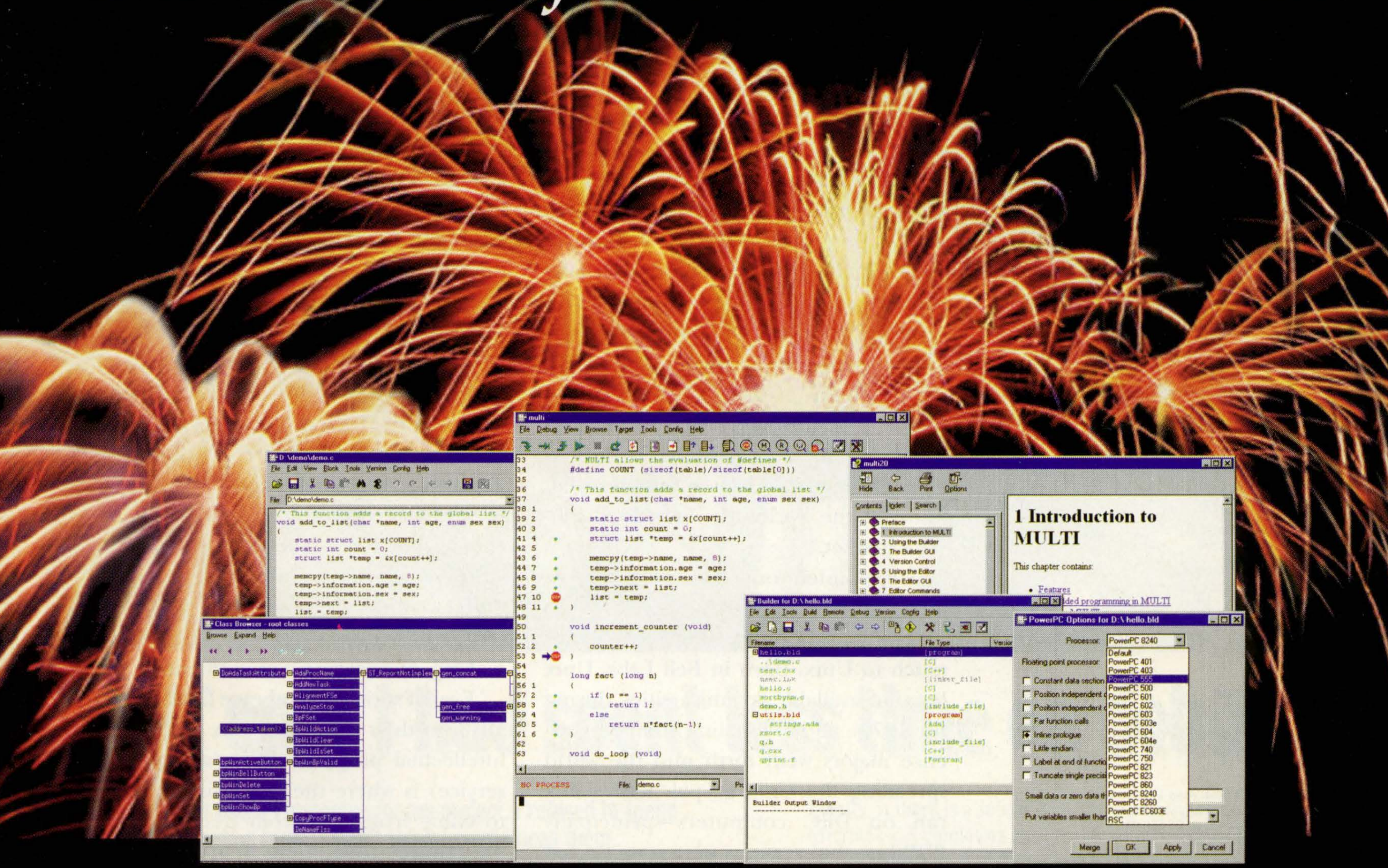
Ironically, because it is open, Linux could diverge, especially embedded and real-time flavors, which could result in several incompatible versions. For Linux to fulfill its potential in the embedded marketplace, vendors must avoid Unix-like battles and must execute the enhancements with an eye toward compatibility. How willing they are to do that is questionable. After all, Linux vendors aren't communists.

Lindsey Vereen

lvereen@cmp.com

MULTI® 2000

The New IDE for the New Millennium



File: D:\demo\demo.c

```
/* This function adds a record to the global list */
void add_to_list(char *name, int age, enum sex sex)
{
    static struct list x[COUNT];
    static int count = 0;
    struct list *temp = &x[count++];

    memcpy(temp->name, name, 8);
    temp->information.age = age;
    temp->information.sex = sex;
    temp->next = list;
    list = temp;
```

```
/* MULTI allows the evaluation of #defines */
#define COUNT (sizeof(table)/sizeof(table[0]))

/* This function adds a record to the global list */
void add_to_list(char *name, int age, enum sex sex)
{
    static struct list x[COUNT];
    static int count = 0;
    struct list *temp = &x[count++];

    memcpy(temp->name, name, 8);
    temp->information.age = age;
    temp->information.sex = sex;
    temp->next = list;
    list = temp;
}

void increment_counter (void)
{
    counter++;
}

long fact (long n)
{
    if (n == 1)
        return 1;
    else
        return n*fact(n-1);
}

void do_loop (void)
```

NO PROCESS   File: demo.c

- Preface
- 1 Introduction to MULTI
- 2 Using the Builder
- 3 The Builder GUI
- 4 Version Control
- 5 Using the Editor
- 6 The Editor GUI
- 7 Editor Commands

1 Introduction to MULTI

This chapter contains:

- Features
- …ded programming in MULTI

Filename	File Type	Version
hello.bld	[program]	
..\demo.c	[C]	
test.cxx	[C++]	
user.lnk	[linker_file]	
hello.c	[C]	
sortbynm.c	[C]	
demo.h	[include_file]	
utils.bld	[program]	
strings.ada	[Ada]	
xsort.c	[C]	
q.h	[include_file]	
q.cxx	[C++]	
qprint.f	[Fortran]	

```
Builder Output Window
------------------------
```

Processor: PowerPC 8240

Default, PowerPC 401, PowerPC 403, PowerPC 555, PowerPC 500, PowerPC 601, PowerPC 602, PowerPC 603, PowerPC 603e, PowerPC 604, PowerPC 604e, PowerPC 740, PowerPC 750, PowerPC 821, PowerPC 823, PowerPC 860, PowerPC 8240, PowerPC 8260, PowerPC EC603E, RSC

Floating point processor:
- □ Constant data section
- □ Position independent
- □ Position independent
- □ Far function calls
- ☑ Inline prologue
- □ Little endian
- □ Label at end of functio
- □ Truncate single precisi

Small data or zero data th

Put variables smaller thar

- New and Improved GUI
- Graphical Browser
- On-Line Help
- Syntax Coloring and Auto Indenting

- Simulator
- Version Control
- C++ Debugging
- EventAnalyzer

- Project Builder
- Code Coverage
- Run-Time Error Checking
- Profiling

The Best Compilers. *The Best Debuggers.*

Now, all wrapped up in a brand new, powerful, easy to use IDE. Once you try MULTI 2000, you won't want to struggle with "old fashioned" tools ever again. Contact us now for a Free Evaluation CD and start the millennium with a bang!

Green Hills
• SOFTWARE INC. •

Tel: 805.965.6044 • Fax: 805.965.6343 • Email: sales@ghs.com • www.ghs.com

Readers make some noise

In reference to Don Morgan's "Pink Noise" column (March 2000, p. 135), generating white noise electronically is one of my many hobbies. I thought readers might be interested to know that one of the methods Mr. Morgan describes (using long shift registers and XORing certain bits) was used by Atari to generate white noise in their arcade games and the 2600 VCS. The arcade game "Asteroids" uses this method to generate the rumble heard during engine thrust and also for explosions, but it was done with individual TTL chips.

While playing with circuits such as these, I've noticed that many of them sound similar, telling me that XOR'ing bits in long shift registers is a common technique with home video games to generate such sounds. The Atari 2600 VCS didn't have much hardware to begin with, so all sounds had to be generated through software. In breadboarding TTL-equivalent circuits, my ears tell me which sounds correspond to shift registers of a particular length.

Don Lancaster discussed this very topic in the February 1989 issue of *Radio-Electronics*. He showed how you could get white noise out of an Apple II computer. His assembly-language program was only 40 bytes long, but it worked beautifully. I used it as a troubleshooting tool for Apple IIe and IIc motherboards as I repaired them for a service depot. I burned that program into an EPROM to help me ascertain how much of the motherboard was working. If the computer instantly gave me a frying noise from the speaker upon power-up, that told me that the microprocessor and associated circuitry was working (it didn't access the RAM at all). If I didn't hear that sound, it helped me zero in on the problem.

William Barden, Jr. explained how to generate sounds on Radio Shack's CoCo in his book *TRS-80 Color Computer Assembly Language Programming*. He, too, used bit-shifting methods and the sounds were exactly like those from the Atari 2600.

Paia Electronics did not use this technique in the 1970s for their electronic music kits, preferring to use reverse-biased transistors to generate white noise. They practically always used the 2N2712. I've found that for this application, a generic equivalent just won't cut it; I get only one volt output on an NTE cross, whereas on a genuine 2N2712, I get eight volts of white noise out, peak to peak, for a particular circuit.

Matt J. McCullar
ARLINGTON, TX

Random concerns

In his column "A Paean to Noise" (Feb. 2000, p. 95), Don Morgan describes the difficulty in creating a truly random random-number generator for the purpose of generating a white noise source. The repeatability and seeding problems are well known issues in such an algorithm. However, for the purposes of spectra, a white noise source usually need not address these concerns. White noise is defined as any process where there is no correlation between a sample and any other in the sequence and is defined only in terms of its second-order moment (variance). The distribution of the sample itself (Gaussian, uniform, and so on) is of no interest spectrally. The negative impact of repeatability and seeding on the randomness of the sequence therefore is also of no interest.

Repeatability will introduce correlation, and the seeding may cause two separate sequences to become correlated. An example of where the second of these issues may become of some importance is when two white noise sources are used in an acoustic application. The sum of the two sequences will not necessarily have the same spectral power as the sum of the powers of the two constituents because they are not uncorrelated random variables.

It appears to me that the issues Mr. Morgan cites would normally be of more interest in a simulation or Monte Carlo analysis.

Mike Henderson
WHITE INSTRUMENTS

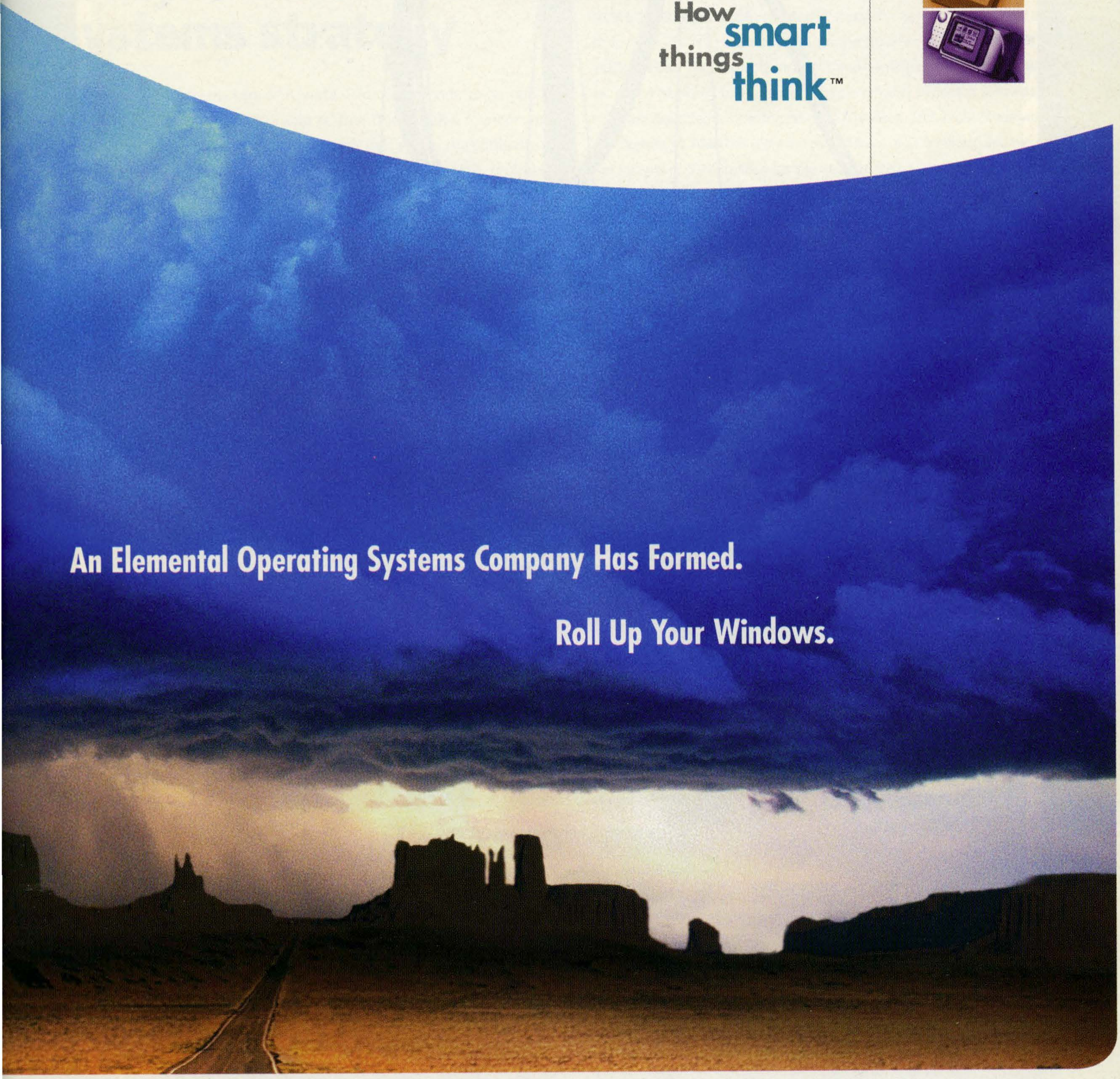
Off-the-shelf security

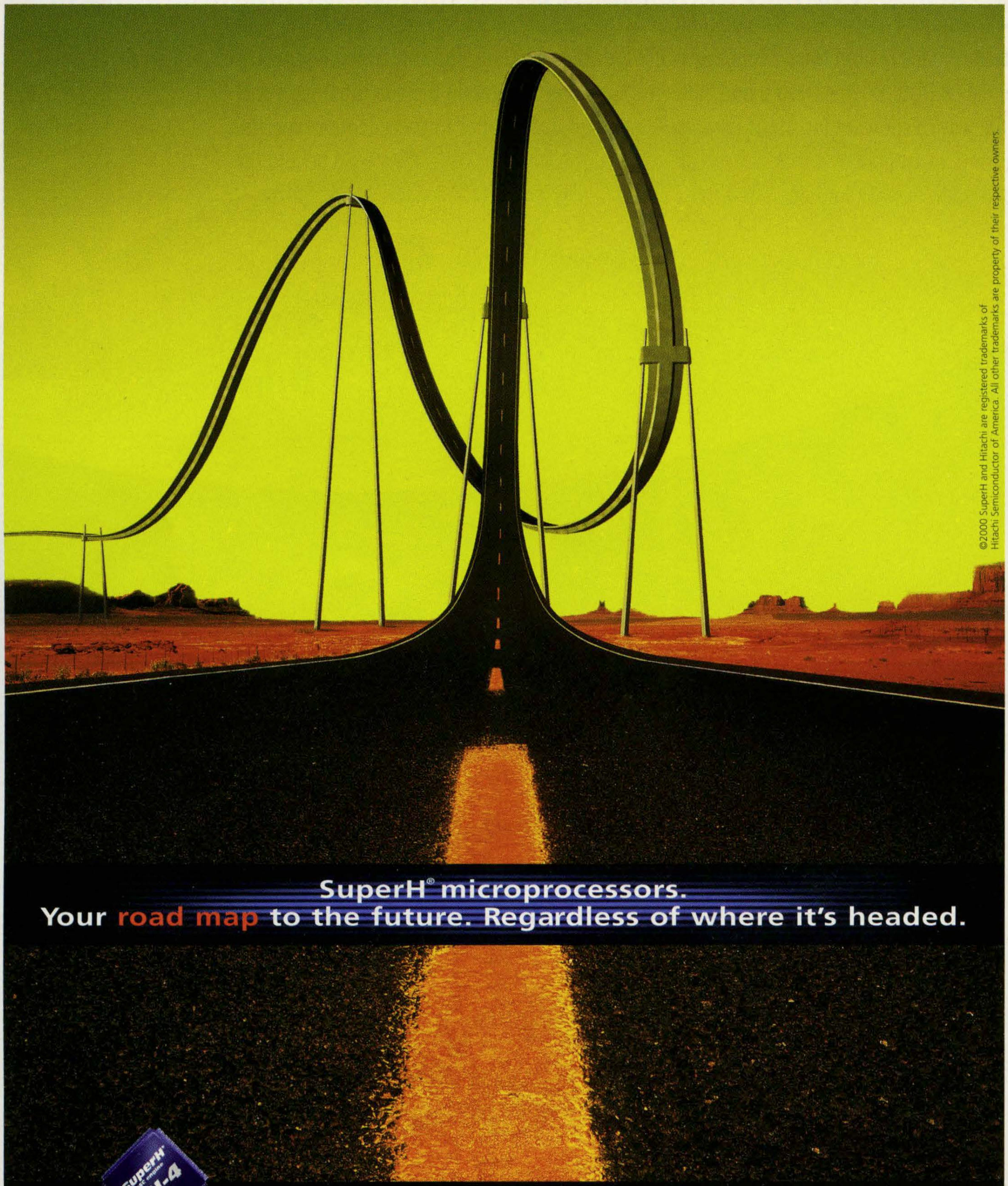
In the article "Protecting Binary Executables" (Feb. 2000, p. 24), Mr Fisher omits mention of several readily available methods of hiding executable code.

The one I am most familiar with is one for the 8051 microprocessor sold by Dallas Semiconductor. The Dallas Semi product comes in two levels of security. The lowest level uses a simple 40-bit encryption of the stored code and addresses, with the key hidden in the processor chip. The higher security level product uses a fancier 64-bit encryption with the key chosen by the CPU, not known to the user, and with stronger defenses against hardware attacks such as probing.

A number of embedded microcontrollers, from Motorola among others, offer EPROM or flash program memory on chip with an option to lock out external access. You don't have to take Mr. Fisher's suggestion of using a custom microprocessor with an unpublished instruction set. Many off-the-shelf security solutions are available.

Byron Blanchard
RATIO METRICS INC.


An Elemental Operating Systems Company Has Formed.
Roll Up Your Windows.


SuperH® microprocessors.
Your road map to the future. Regardless of where it's headed.

QNX announces likely IPO and open systems strategy

QNX Software Systems Ltd. of Kanata, Ontario, intends to go public in "about eight months," according to Dan Dodge, QNX founder and chief technical officer. In conjunction with that move, the company is seeking a new president to help with the transition. "We probably should have done this a year ago, but staying private has helped us work on our long-term plans," Dodge said.

Those plans include an open systems strategy to target the "e-device" market and to provide support for the Linux operating system. Developers will be able to download the QNX real-time platform free for non-commercial use, and the company will release source code for many QNX applications, drivers, and libraries. In addition, QNX Software has integrated a high level of Linux compatibility into its new platform, which, according to Dodge, means binary compatibility with Linux. Developers can pre-register for their copy of the QNX real-time platform at *http://get.qnx.com.* It will be available midsummer.

The QNX real-time operating system is the fifth ranked RTOS used by *Embedded Systems Programming* readers for 16- or 32-bit development, according to a recent survey.

Alchemy Semi enters e-device market

Alchemy Semiconductor Inc., an Austin, TX-based fabless semiconductor company targeting the Internet device market, recently announced a $15 million first round of venture capital financing from US Venture Partners, Austin Ventures, and Telos Ventures.

Alchemy Semiconductor will use the money to complete development and marketing of its first product and to create a corporate infrastructure as the company spins off from Cadence Design Systems, according to a press release. Cadence provided seed money for the microprocessor design firm in 1999 when Alchemy was founded by several members of the former StrongARM design team. Alchemy plans to make its first product announcement at the Embedded Processor Forum in San Jose, CA, June 12-16. Alchemy will continue to utilize Cadence's EDA software tools. Cadence will remain an Alchemy shareholder after the spin off.

The company has also named Eric Broockman president and CEO.

THE NUMBERS GAME

BP MICROSYSTEMS INC. announced first quarter revenue of $10.5 million, an increase of 48% over last year's numbers. ● **STMICROELECTRONICS** reported first quarter net revenues of $1.7 billion, a 15.2% increase over year-ago numbers. ● **MICROCHIP TECHNOLOGY INC.** expects net sales and earnings for the fourth quarter ended March 31 to be stronger than previously expected. Record net sales are expected to be in excess of $140 million, 36% higher than year-ago figures. ● **AMD** has reported record sales of $1.09 billion and record net income of $189 million for the quarter ended April 2. ● **CYPRESS SEMICONDUCTOR CORP.** announced revenue of $207.9 million for the fourth quarter of fiscal year 1999. ● **RADISYS CORP.** has reported revenues of $72.9 million for the quarter ended Dec. 31, 1999, a 52% jump from last year's numbers. ● **I-LOGIX INC.** has received $10.5 million in a mezzanine round of venture financing. ● **LINEO INC.** has announced the receipt of $37 million in investment money, mostly from strategic investors.

Briefly Noted...

Green Hills Software's MULTI 2000 Integrated Development Environment is available for the StarCore SC100 generation of digital signal processors. ✳ The Linux operating system has been ported to **NEC**'s standard evaluation platforms for its VR4121, VR5432, and VR5000 64-bit MIPS RISC microprocessors. ✳ **Lineo Inc.** has announced that it will offer embedded Linux support for the MIPS architecture. ✳ **Applied Microsystems** and **MontaVista Software Inc.** have partnered to integrate Applied Microsystems' CodeTEST development tool and MontaVista's HardHat Linux. ✳ **The Ada Resource Association** (ARA) has announced that Ada is the first language to standardize a process for testing compilers' implementations. ✳**CPU Tech** won a development and production program to modernize a critical radar processing system on F-16 fighter planes. ✳ **Traquair Data Systems Inc.** and **Strategic Partner Hunt Engineering Ltd.** now support Linux with their HERON and TIM-40 multiple-processor DSP boards. ✳ **BOPS Inc.**'s Technology Evaluation License program was designed to help customers evaluate BOPS' DSP cores for high-volume system-on-chip applications. ✳ The USNET Embedded TCP/IP Protocol Suite, from **U S Software**, has been ported to **Express Logic Inc.**'s ThreadX real-time operating system. ✳ **Invensys Network Systems** and **emWare Inc.** have agreed to jointly develop and market a ControlServer which will control devices from user interfaces in a home or business and from outside locations via the Internet. ✳ **Lineo Inc.** has announced strategic relationships with the following Taiwanese embedded systems manufacturers: **Arima Computer Corp.**, **Compal Electronics Inc.**, **First International Computer Inc.**, **Micro-Star International**,

Microsoft reorganizes embedded group

Microsoft Corp. recently announced a structural reorganization involving its embedded systems group. The company also announced that it will focus efforts on Windows CE v. 3.0, scheduled to ship in June.

Bill Veghte, vice president of the new embedded and appliance platform group, is spearheading both the Windows CE and the Embedded Windows design groups. Windows CE 3.0 will boast improved real-time features, an Internet Explorer 4.0 browser, and will support 256 priority levels and nested interrupts. Veghte's goal is to offer embedded platforms that can be tailored to any customer configuration or level of integration.

Metrowerks joins Embedded Linux Consortium

Metrowerks has announced that it is a founding member of the Embedded Linux Consortium. The Consortium, formed at the recent Embedded Systems Conference in Chicago, is a coalition of embedded software companies and engineers to promote and implement the Linux operating system in embedded applications.

Metrowerks will join IBM, Hewlett-Packard, Motorola, 3Com, RedHat, and others to help define the charter of the group and collaborate with other members on efforts to expand the acceptance of embedded Linux, according to a news release.

Information about the Embedded Linux Consortium can be found at *www.embedded-linux.org.*

Viewlogic and Summit Design are now Innoveda

Viewlogic Systems Inc. and Summit Design Inc. announced completion of their merger and are now Innoveda Inc. The new company will market software and services targeting engineers developing electronic systems for telecommunications, transportation, computers, and consumer electronics. Innoveda trades under the Nasdaq symbol INOV.

The merger was approved on March 20 by shareholders of both companies. It was structured as the acquisition of Viewlogic by Summit. Approximately 16.2 million Innoveda shares were issued in exchange for all outstanding shares of Viewlogic. Innoveda also assumed Viewlogic stock options, exercisable for up to approximately 2.0 million Innoveda shares.

NAMES IN THE NEWS

FARZAD ZARRINFAR has joined Altera Corp. as vice president of product marketing. ● NETsilicon Inc. has named **ERIC A. KRAIESKI** vice president of product marketing and **STEPHEN E. MARTIN** director of marketing communications. ● **PAUL H.F. VROOMEN** has been appointed president and COO of SandCraft Inc. ● Linux NetworX Inc. has named **STEPHEN HILL** vice president of business development.

Briefly Noted (cont.)...

MITAC International Corp., and **Wisecom**. ✳ **Motorola Inc.** will acquire **C-Port Corp.**, a network processor developer, for approximately $430 million. ✳ **Extended Systems** has licensed its Bluetooth software technology to **3Com Corp.** for use in Palm Inc. products. ✳ **Lineo Inc.** will acquire **Rt-Control Inc.**, makers of the uClinux operating system for microcontrollers. ✳ **Ultima Communication** and **Silicon Wave** have announced a cross-license agreement for protocol software for the implementation of Bluetooth wireless technology. ✳ **ParaSoft** has made its error-prevention and error-detection tools available for embedded systems development. ✳ **Applied Microsystems Corp.** has signed an agreement with **STMicroelectronics** to license Applied's CodeTEST-Software-in-Circuit software verification tools. ✳ **Centura Software Corp.** recently went live with its first open source data management solution for information appliances. Centura's db.linux is available at *www.openavenue.com/db.linux.* ✳ **Lineo Inc.** has partnered with **MIPS Technologies** to provide embedded Linux support for the MIPS architecture. ✳ **MIPS Technologies** and **Microware Systems Corp.** have partnered to provide the OS-9 real-time operating system for 32- and 64-bit MIPS processors. ✳ **Lynx Real-Time Systems Inc.** announced the delivery of **Hewlett-Packard**'s ChaiVM embedded virtual-machine technology on the Lynx BlueCat Linux operating system. ✳ **Telelogic AB** will acquire the COOL:Jex business from **Sterling Software**. ✳ **Mentor Graphics Corp.** and **Telelogic** announced the release of an integrated hardware/software solution linking Mentor's Seamless Co-Verification Environment (CVE) and the Telelogic Tau SDL Suite. ✳ **Arca Technologies** has announced its new WaveCatcher protocol analyzer for Bluetooth. **Wind River Systems** will acquire Calgary, Alberta-based **AudeSi Technologies Inc.**, a supplier of embedded Java-based tools, for 1.075 million shares of Wind River stock. ✳ **OpenTV** and **Spyglass Inc.** have announced that they will merge. The transaction is valued at approximately $2.5 billion. ✳ **DDC-I** has introduced its SCORE (Safety Critical, Object-oriented, Real-time Embedded) integrated development environment.

Some of the biggest names in the business come to us
for smart networking solutions. We're glad to chip in.

Looking for a smart solution for your next-generation router, switch, server, modem, or desktop device? There's only one name to know: DigitalDNA from Motorola. DigitalDNA chips, systems, software and ideas — embedded solutions that help smart companies create smart products. And you'll find it in leading-edge technology like 0.10μm copper interconnect CMOS-based solutions. The world's fastest copper interconnect SRAMS. And a wide portfolio of highly scalable PowerPC cores. So if you're ready to go big time with your networking or computing design, we're happy to help.

www.digitaldna.motorola.com

Digital DNA™
from Motorola
THE HEART OF SMART.™

Now Accepting Applications.

Publish e-books and develop programs for a brand-new format – with more screen, more resources and more security.

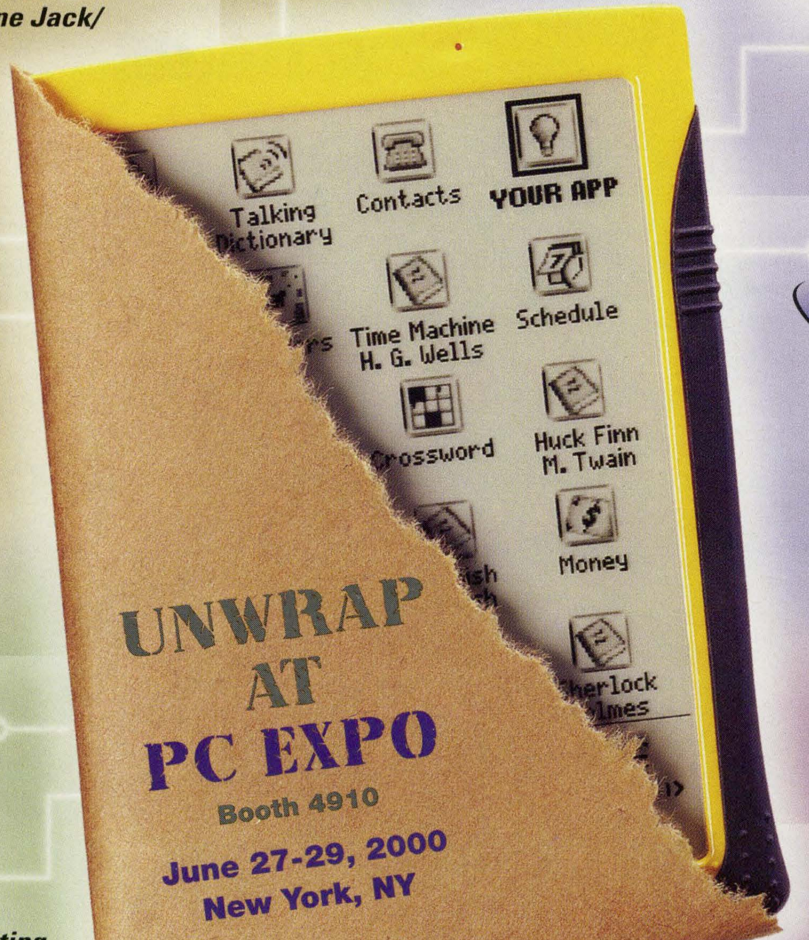
Microphone/ Headphone Jack/ Speaker

C++ Development & Debugging

MMC (Multimedia Card) Slot

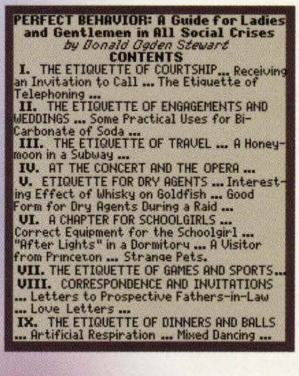
8-16 MB of RAM Built In

Natural Handwriting Recognition



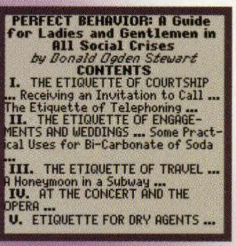
USB Port

LARGE Screen with 87% more pixels



PERFECT BEHAVIOR: A Guide for Ladies and Gentlemen in All Social Crises
by Donald Ogden Stewart
CONTENTS
I. THE ETIQUETTE OF COURTSHIP... Receiving an Invitation to Call ... The Etiquette of Telephoning ...
II. THE ETIQUETTE OF ENGAGEMENTS AND WEDDINGS ... Some Practical Uses for Bi-Carbonate of Soda ...
III. THE ETIQUETTE OF TRAVEL ... A Honeymoon in a Subway ...
IV. AT THE CONCERT AND THE OPERA ...
V. ETIQUETTE FOR DRY AGENTS ... Interesting Effect of Whisky on Goldfish ... Good Form for Dry Agents During a Raid ...
VI. A CHAPTER FOR SCHOOLGIRLS ... Correct Equipment for the Schoolgirl ... "After Lights" in a Dormitory ... A Visitor from Princeton ... Strange Pets.
VII. THE ETIQUETTE OF GAMES AND SPORTS...
VIII. CORRESPONDENCE AND INVITATIONS ... Letters to Prospective Fathers-in-Law ... Love Letters ...
IX. THE ETIQUETTE OF DINNERS AND BALLS ... Artificial Respiration ... Mixed Dancing ...

Typical PDA Screen



PERFECT BEHAVIOR: A Guide for Ladies and Gentlemen in All Social Crises
by Donald Ogden Stewart
CONTENTS
I. THE ETIQUETTE OF COURTSHIP ... Receiving an Invitation to Call ... The Etiquette of Telephoning ...
II. THE ETIQUETTE OF ENGAGEMENTS AND WEDDINGS ... Some Practical Uses for Bi-Carbonate of Soda ...
III. THE ETIQUETTE OF TRAVEL ... A Honeymoon in a Subway ...
IV. AT THE CONCERT AND THE OPERA ...
V. ETIQUETTE FOR DRY AGENTS ...

Get With the Program!

Large screen, latest technologies and **amazing low retail prices** - they all add up to more customers for your new e-books and apps. Develop for the world's first open and secure system. Discover co-marketing opportunities with the one company that's sold over 21 million electronic books in 45,000 retail outlets worldwide and on the Web.

Register now for your FREE SDK at www.franklin.com/devzone

Franklin®
Electronic Publishers

Dan Saks

Is an Array Really Just a Pointer?

Without a doubt, arrays are among the most confusing aspects of C. Arrays in C++ are essentially the same as they are in C, and C++ programmers seem to misunderstand arrays at least as often as C programmers do. Maybe even more.

Now, I'm not suggesting that C and C++ programmers don't know how to use arrays. Most do, at least in common, everyday applications involving arrays with a single dimension. Unfortunately, too many programmers cope with unusual situations by resorting to trial and error. At some time or another, nearly all of us have dealt with a compiler error message by using what I call the "Oh, what the heck" approach: you throw a star or parentheses somewhere into the code and hope it will compile.

Most of us learn to program by mapping the constructs we see in the source code into a conceptual model of what's going on in the compiler and the runtime system. Often we start with a model that's too simple and we have to refine it as we learn more about computing. As professional programmers, our model ought to become pretty accurate after a while. Unfortunately, there's something about arrays in C and C++ that lets even competent programmers cling to inaccurate programming models for a remarkably long time.

"So what?" you ask. After all, if your programming model holds up for all the common applications of arrays, what difference does it make if you have to resort to "Oh, what the heck" once in a while? As much as it pains me to admit, the "Oh, what the heck approach" often works pretty well. However, when it fails, it fails big time. The code compiles and links all right, but then it produces extremely subtle run-time bugs.

Any C/C++ compiler will allow you to interchange arrays and pointers in your programs. Underneath it all, though, they are indeed different entities.

Misunderstanding arrays and pointers is probably more of an issue for C++ programmers than for C programmers. C++ supports function name overloading, operator overloading, and templates, all of which rely on elaborate compile-time analysis of data types. These features, and the compiler error messages that go with them, often force programmers to confront what they don't quite understand about the nature of arrays and pointers.

This month, I'll take a stab at clearing up some of what I've found to be common misunderstandings about arrays.

Source of the confusion

By themselves, arrays in C aren't any more mystifying than they are in other programming languages. Most C programmers learn about arrays before they learn about pointers, so they get to use arrays for a little while before realizing they don't understand them. Real confusion usually doesn't set in until they learn about pointers and see just how blurry the line is between arrays and pointers.

For example,

```
T x[N];
```

declares `x` as an array with `N` elements of type `T`, where `N` is some previously declared integer constant and `T` is some previously declared type. The elements in the array have subscripts or indices from 0 to `N`–1, inclusive. You can refer to the 0th element of `x` by using the subscripting or indexing expression `x[0]`. For any integer expression `i`, you can refer to the *i*th element of `x` as `x[i]`.

On the other hand, you can also treat `x` as a pointer and refer to the 0th element of `x` as `*x`, which is a pointer indirection or dereferencing expression. For any integer expression `i`, you can refer to the *i*th element of `x` as `*(x+i)`, which combines pointer arithmetic with indirection.

In addition,

```
T *p = x;
```

defines p as a pointer to type T, whose initial value is the address of the 0th element of x. You can then refer to the elements of x as the "elements pointed to by p" using expressions of the form `*(p+i)`. You can even refer to the elements of array x as if they were the elements of array p by using expressions of the form `p[i]`.

In short, since there are so many ways that you can use an array as if it were a pointer, does that mean an array is really just a pointer? Or is it something else?

A short quiz

Before I give you the answer to those questions, let me rephrase them a little more precisely.

Suppose you're developing or maintaining a C or C++ compiler. You feed source code to the compiler containing the declaration:

```
int i;
```

Using the debugger, you stop the compiler just after it reads past the end of the declaration. You look in the compiler's symbol table to see how the compiler recorded the declaration. You should expect to see information in the table that describes i as an object of type int.

Now suppose you feed the declaration:

```
int *p;
```

to the compiler. Again, you use the debugger to look in the symbol table just after the compiler reads past the declaration. What should you expect to see recorded as the description of p? You should expect to see that p is an object whose type is "pointer to int."

Okay, now here's the question for you. You might want to cover the next two paragraphs so you don't see the answer until you've thought about it for a moment.

What does the compiler record for the following declaration?

```
T x[N];
```

Yes, I know we say that x is an "array of N elements of type T," but what type does the compiler actually store in the symbol table as the type of x? Is x really an array, or is it just a pointer?

The answer is...

x really is an "array with N elements of type T."

Arrays really are arrays

The fact is, there really are objects with array types in C and C++. When you declare `x` as:

```
T x[N];
```

it really is an array. It's not a pointer. Not in the sense that `p` declared as:

```
T *p;
```

really is a pointer.

How, then, does an assignment such as:

```
p = x;
```

compile? When it evaluates the expression, the compiler treats `x` as if it were a "pointer to `T`" whose value is the address of the 0th array element. However, the compiler treats `x` as a pointer only for a moment, just long enough to assign the pointer value of `x` to `p`. All the while, the compiler's symbol table says that `x` is an array.

Momentary conversions such as this occur frequently during expression evaluation. For example, given:

```
double d;
int i;
```

we often say that the expression `d + i` "converts `i` to `double`" before adding `d` and `i`. However, the program doesn't really change `i` into a `double`. Rather, it creates a temporary `double` object (possibly in a register) initialized with the value of `i`, and then adds `d` and the temporary. The temporary vanishes soon thereafter. Object `i` remains an `int`.

Subscripting expressions such as `a[i]` also treat arrays as pointers. Both C and C++ expect `a` to be an expression of some pointer type. If `a` is a pointer, it remains as such. If `a` is an array, the compiler converts `x` to a pointer before compiling the rest of the expression. Once again, the compiler doesn't actually change `a` into a pointer; it creates a temporary pointer object whose value is the address of the initial element in `a`.

If the compiler treats the array as a pointer every time you use an array in an expression, how can you ever tell that it is an array? There is at least one context where the array-to-pointer conversion does not occur—in a `sizeof` expression. For example, given:

```
int v[10];
```

then `sizeof(v)` yields `sizeof(int) * 10`, not `sizeof(int *)`.

Another context in which the array-to-pointer conversion does not occur is when an array is the operand of the unary `&` (address-of) operator. For example, given:

```
T x[N];
```

the expression `&x` yields a result of type "pointer to array with `N` elements of type `T`," not "pointer to pointer to `T`." You can tell because:

```
T (*p)[N] = &x;     /* yes */
```

compiles without error. On the other, if it doesn't generate an error, then:

```
T **q = &x;        /* no */
```

generates at least a warning that this involves a questionable pointer conversion.

Arrays and pointers blur together in other ways. I'll try to clear them up in the coming months. **esp**

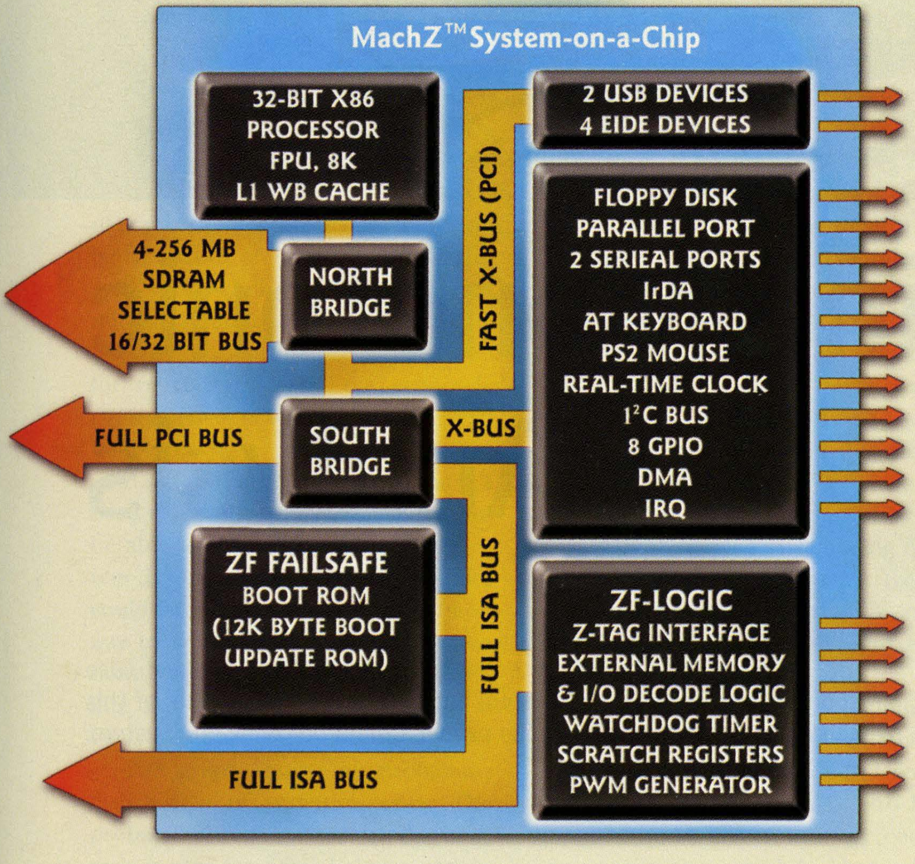
Dan Saks is the president of Saks & Associates, a C/C++ training and consulting company. He is also a consulting editor for the C/C++ Users Journal. *He served for many years as secretary of the C++ standards committee and remains an active member. With Thomas Plum, he wrote* C++ Programming Guidelines. *You can write to him at dsaks@wittenberg.edu.*

Finally
a real x86
ZF
LINUX DEVICES
MachZ™ x86 PC
3100-0200-00
123456789012
©1999
Phoenix Technologies
FailSafe
EMBEDDED
actual size
This image can now be taken Literally.

Introducing the "FailSafe MachZ" for the next generation of embedded systems.

System on a Chip

You can design the MachZ system into virtually anywhere. It's very small and actually runs cool to the touch. Think about it: you could embed this little power house in a system that is completely sealed. Consider the possibilities, consider the facts. Then put your engineering brain to work.



MachZ™ System-on-a-Chip

32-BIT X86 PROCESSOR FPU, 8K L1 WB CACHE

4-256 MB SDRAM SELECTABLE 16/32 BIT BUS

NORTH BRIDGE

FAST X-BUS (PCI)

FULL PCI BUS

SOUTH BRIDGE

X-BUS

ZF FAILSAFE BOOT ROM (12K BYTE BOOT UPDATE ROM)

FULL ISA BUS

2 USB DEVICES
4 EIDE DEVICES

FLOPPY DISK
PARALLEL PORT
2 SERIEAL PORTS
IrDA
AT KEYBOARD
PS2 MOUSE
REAL-TIME CLOCK
I^{2}C BUS
8 GPIO
DMA
IRQ

ZF-LOGIC
Z-TAG INTERFACE
EXTERNAL MEMORY
& I/O DECODE LOGIC
WATCHDOG TIMER
SCRATCH REGISTERS
PWM GENERATOR

Ultra Low Power <500mW@100MHZ

Auto-Boot FailSafe System allows crash-immune Internet upgrades - FailSafe Boot, Flashless Operation, Dual WD-Timer, Z-tag 2M-bit/s Flash download, Scratch Registers

Fully PC Compatible runs all x86 code natively

Easily Interface HomePNA, Ethernet, Modems, Graphics Controllers, PC/104, MPEG, XDSL, ISDN, IrDA, Flash, CAN, GPS

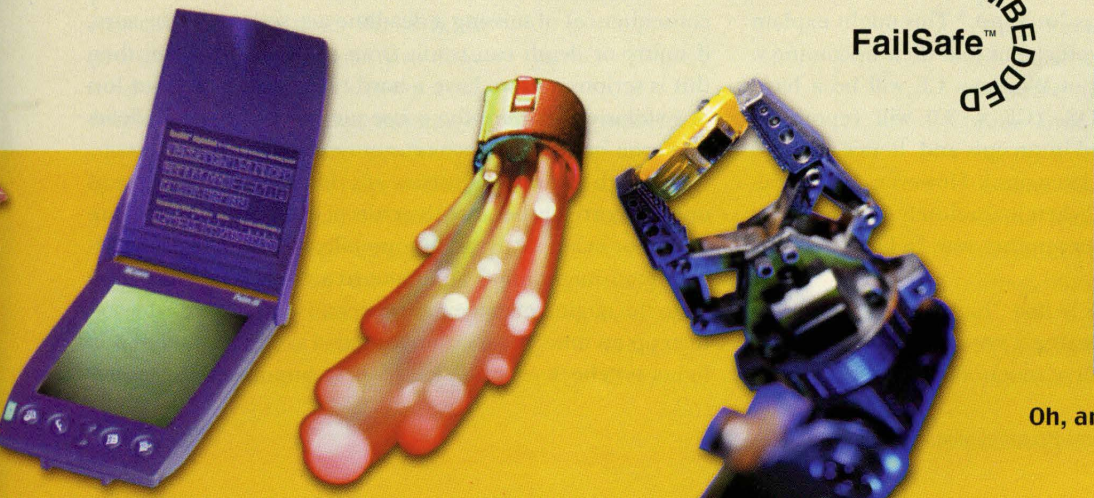
Includes BIOS and Linux O/S or WindRiver VxWorks RTOS

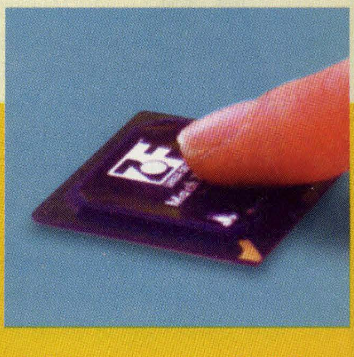
phoenix™ technologies

ZFLINUX DEVICES
Tel:800.683.5943
www.zflinux.com

ZF™

FailSafe™ EMBEDDED

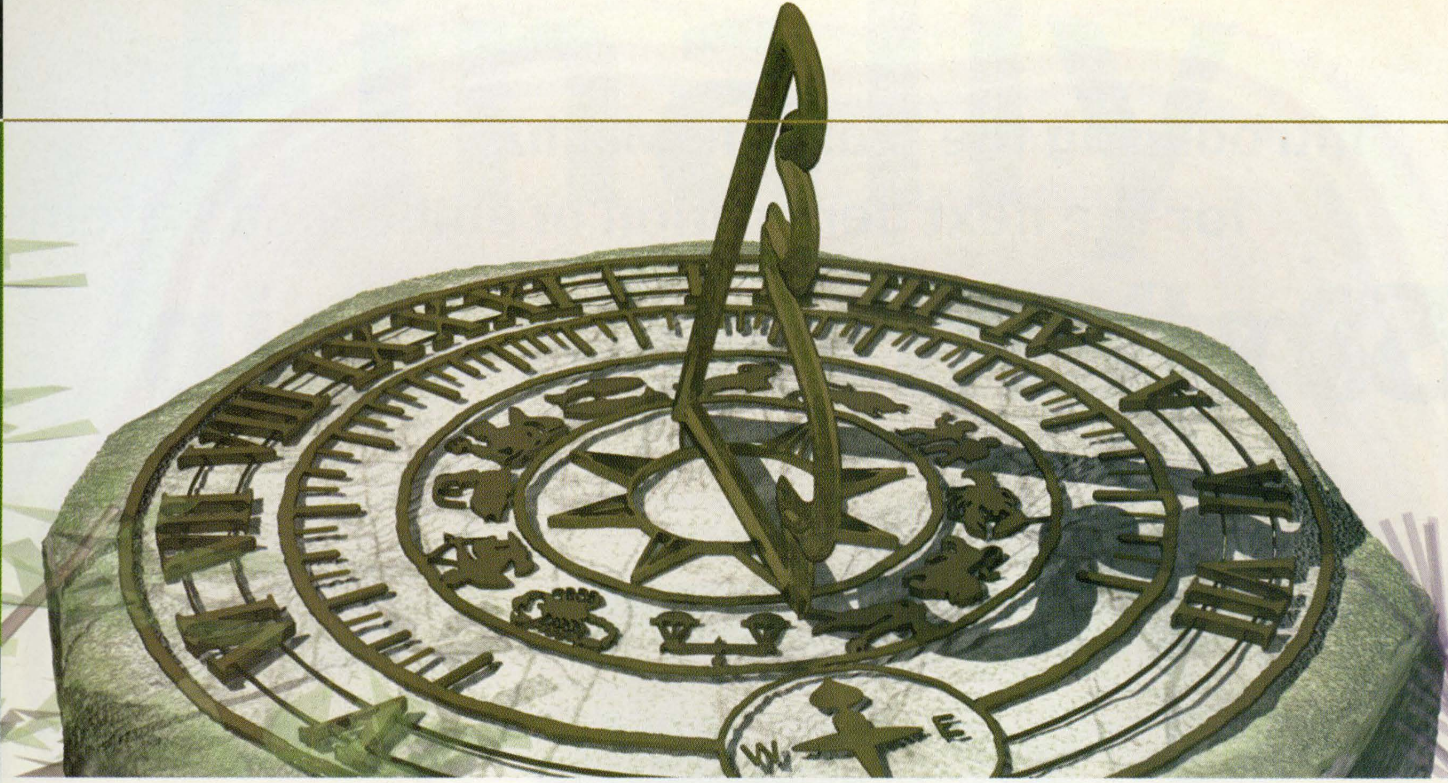




Oh, and did we mention that it runs c

388 BGA Runs at a cool 1/2 Watt
not a typical 8 to 10 Watts

KEN TINDELL



Deadline Monotonic Analysis

This article presents a technique for analyzing the worst-case response times of tasks in a system and shows how the results of this analysis can be used to ensure the proper timing behavior of a system.

A common misconception is that "hard real-time" means "very fast." This might explain why Microsoft claims that their upcoming v. 3.0 release of Windows CE will be a hard real-time OS. (CE v. 3.0 will reportedly allow nested interrupts and, hence, achieve sub-millisecond interrupt latencies.) However, hard real-time is better described as "when it absolutely, positively has to be done on-time." Or, put another way, "a late result is a wrong result."

Another misconception is that "hard real-time" means "safety critical" and that because a system cannot kill someone there's no need to worry much about meeting deadlines. However, a hard real-time system is one where the consequences of missing a deadline are serious. To be sure, if injury or death can result from a missed deadline then this is serious and we have a hard real-time system. But lots of systems exist where the consequences of missing a deadline even occasionally are economically unacceptable.

In high-volume industries it's important that systems operate correctly because the costs of even trivial failures are high. For example, in an automobile, the instrument panel computer might apply a timeout to a regular RPM message from the engine management system. If the message doesn't turn up on time, the instrument panel assumes a failure and lights a "check engine" lamp. It is important to do this

It's clear that if we want to build a hard real-time system then we have to be able to determine, before the system is deployed, whether deadlines could be missed or not.

because if the engine management system isn't talking properly on the network then there might be some serious faults, and the driver should take the automobile to a workshop to check for a wiring fault. But if the vehicle systems haven't been designed to meet hard deadlines, and occasionally the RPM message turns up a little late, the time-out will trigger and the garage mechanics will find no actual fault. If this kind of overrun occurs even once in a million hours then, with the millions of hours of operation of a particular vehicle type every day, there will be thousands of "no fault found" warranty claims each year. This could cost millions of dollars, not to mention the damage to a corporation's reputation for reliability. Clearly, the economic consequences of missing deadlines can be severe.

It's clear that if we want to build a hard real-time system then we have to be able to determine, before the system is deployed, whether deadlines could be missed or not. One common way to try to find out the worst-case timing behavior of a system is via testing. But this is a poor technique, as Figure 1 shows.

The graph shows the probability distribution of response times. The probability of a very short response time is zero for times below a certain point. This point corresponds to the best-case response time. Similarly, the probability again becomes zero above a certain response time: the worst-case response time. In testing, the longest observed response time generally is less than the worst-case response time. This means that a deadline might be deemed "always met" but, in fact, could be missed after the system is deployed. A high-profile case of this is when the NASA JPL Pathfinder probe first reached Mars: although the control software passed its testing on Earth, a deadline was missed on Mars and the control system crashed. We'll look at this case in more detail later.

Static scheduling techniques

We can see how important it is to meet hard real-time deadlines. Historically this has been done by static scheduling techniques. Figure 2 shows a system scheduled off-line.

In Figure 2 there are five things to do, each at its own rate. In the example system there is a requirement that each activity (or "task") is completed before it is due to run again. The system is scheduled by building a 4ms schedule composed of four 1ms time slots. As Figure 2 shows, the highest rate task (with a period of 1ms) appears in each slot. Lower rate tasks appear in just one or two of the slots. The schedule allows for time due to handling interrupts by calculating the largest number of interrupts from each source that could occur in any 1ms interval, and then allowing for this time in all the slots.

This kind of static cyclic scheduling provides superb real-time predictability, and doesn't need any kind of real-time operating system (RTOS) to be implemented. The following code implements the schedule shown in Figure 2:

```
main()
{
  do_init();

  for(;;) {
    busy_wait();
    t1();
    t2();
    busy_wait();
    t1();
    t3();
    t5();
    busy_wait();
    t1();
    t2();
    busy_wait();
    t1();
    t4();
  }
}
```

This pre-scheduled approach has some major problems. One problem is inefficiency: the system does not maximize the use of the processor. One reason is that urgent but infrequent events are handled very poorly. An event needs to be polled because it's never sure in which sub-schedule an event will occur, and so the schedule needs to poll frequently for each urgent event to be sure of responding on time. This means that a lot of processing time must be reserved for handling an event, even though in reality that time is hardly ever used for hard real-time processing. The situation with interrupts is similar: in every slot we have to reserve time for handling interrupts even though those interrupts rarely occur.

The static approach is also inefficient because of the way task periods have to fit neatly together to keep the schedule cycle time short. In the schedule shown in Figure 2, we have periods of 1ms, 2ms, and 4ms. If we wanted to add a new task with a period of 5ms we would either have to increase the overall schedule cycle time from 4ms to 20ms or else we would have to change the task period to 4ms. Since the schedule cycle time is the least common multiple of the periods of the tasks, the schedule soon grows out of control.

Task periods are usually shortened to neat values that fit a short schedule. Shortening a task's period means that

RUPERT ADLEY

[With static scheduling], a lot of processing time must be reserved for handling an event, even though in reality that time is hardly ever used for hard real-time processing.

FIGURE 1 Response time vs. probability curve

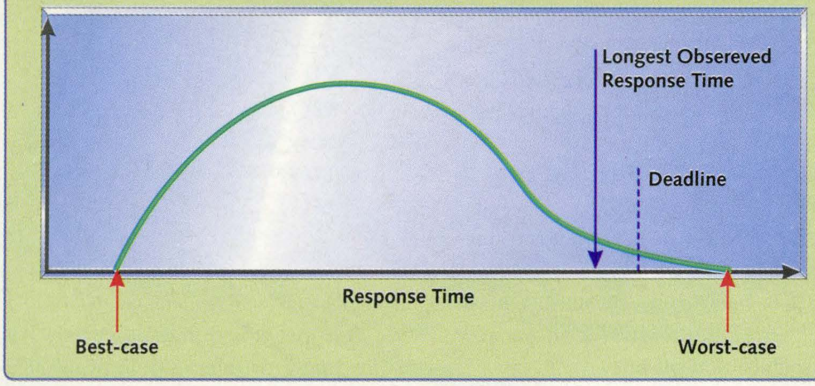
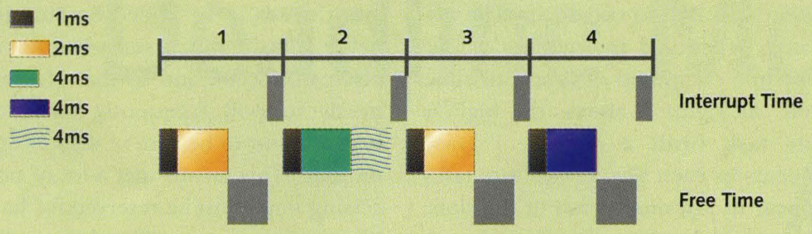


FIGURE 2 A statically scheduled system



the processor load due to the task is increased. For example, if a task has an execution time of 500µs and its period is shortened from 5ms to 4ms, then the processor load due to the task is raised from 10% to 12.5%, a waste of 2.5% of the processing resources. The overall wastage can be high, but this is often not noticed because the implementation technique is assumed right back when the system is first specified and the task periods are set as neat periods.

A major problem with the static approach is the difficulty of writing and maintaining application code. Figure 2 shows how the spare time is allocated between the time slots. The problem is that, although there might be quite a lot of spare time across the schedule, there's not enough spare time in any one place; a task has to fit within the spare time in a single time slot. When a task executes for longer than the spare time in any slot, there must be a restructuring of the task source code. The code must be broken down into two or more sub-tasks. These must be called from different time slots to use the available spare time. Unfortunately, it's usually not trivial to split a piece of code so that the execution time of each piece is neatly balanced. In any case, additional code is usually needed to save the internal state at the end of one sub-task into global variables and re-load the state from these global variables in the subsequent sub-task. This wastes valuable RAM, ROM, and processing time.

Later in the development or maintenance phase a sub-task might itself need to be broken down into sub-tasks to re-allocate processing time between time slots. As the development progresses, the logical structure of the application tends to become corrupted, and it becomes harder and harder to maintain the code. As well as increasing software life-cycle costs, this poor management of complexity can have a severe effect on the ability to develop quality software within time-to-market targets.

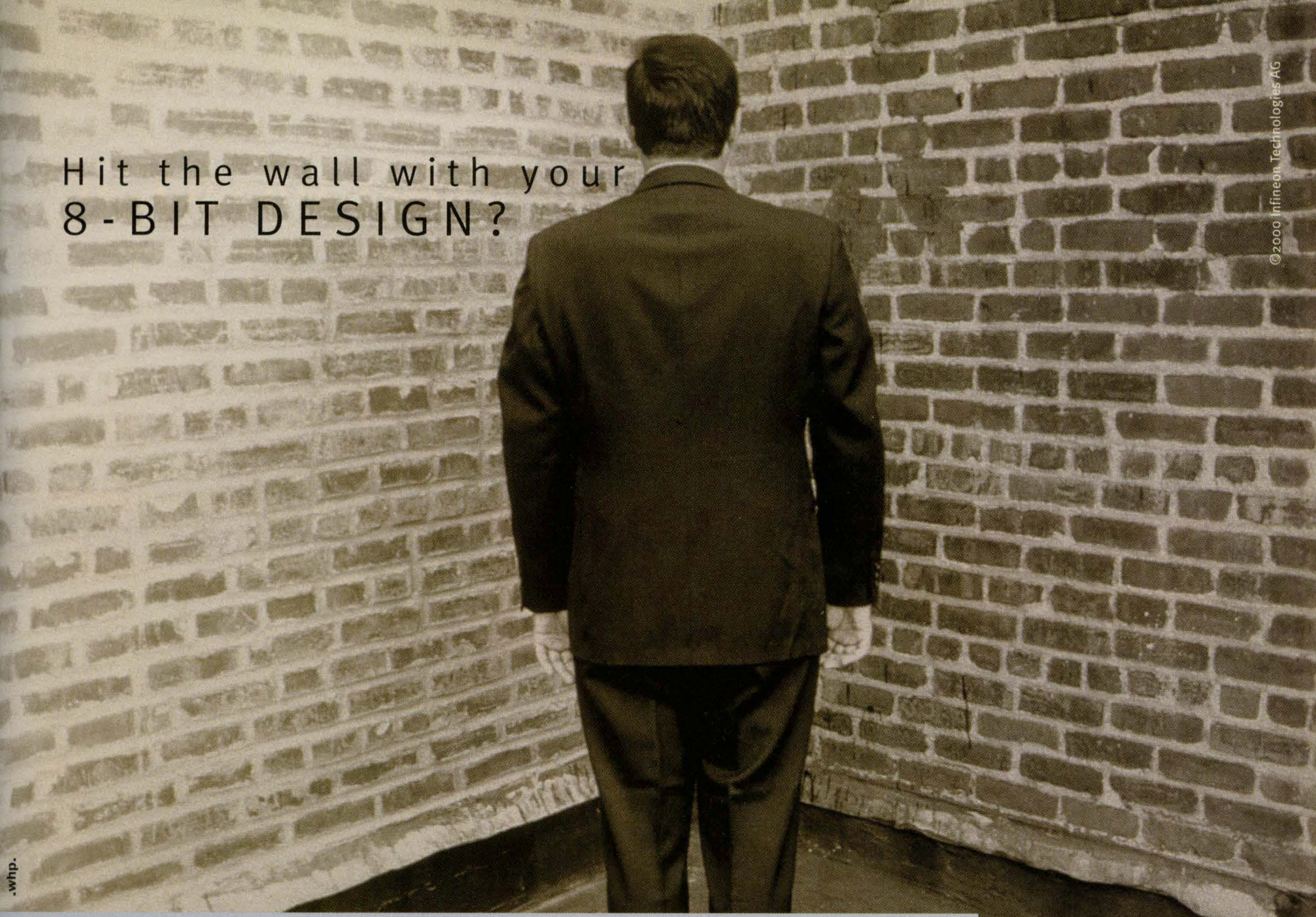
Priority-based scheduling

There is a better way of constructing a hard real-time system: the use of multitasking under an OS scheduling tasks by priorities.

This kind of system has several tasks, each an independent thread of control and each assigned a priority. The OS ensures that at any given time the highest priority task that's ready is actually running. If a task becomes ready to run and this task is of higher priority than the current task, the OS suspends the current task and starts running the new task. If a task is running and suspends itself (waiting for an event or for an interval of time, for example) then the task is no longer ready to run and the OS resumes executing the next highest priority task.

Priority-based scheduling is potentially more efficient than static scheduling techniques for two reasons. First, selecting task periods based on harmonic values is unnecessary. So if the application requires 5ms, 12ms, and 7ms periods then the OS can run these tasks at those rates and avoid running tasks at higher rates. Of course, this benefit is often overlooked because in many systems, the specification assumes that the implementation will use static scheduling and harmonic periods are specified. A second benefit is that each urgent event is handled by a high priority task that is made ready only when the event occurs. This gives a short response time to the event and uses the processor only when necessary.

Fortunately, priority-based scheduling is a popular implementation technique. In a survey by *Real-Time Magazine* (issue 97-3), 41 of the more


Hit the wall with your
8-BIT DESIGN?

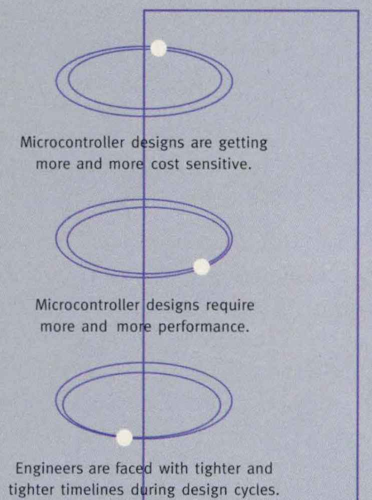
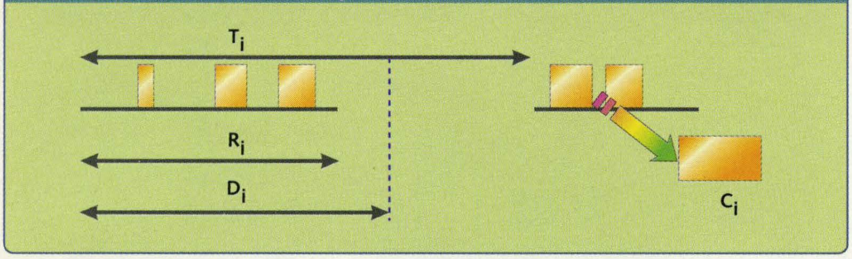


FIGURE 3 Notation for analysis



than 50 RTOSes listed supported fixed priority scheduling.

Analysis of priority-based systems

Although the system can be constructed and maintained more easily than a static approach, and although the processor can be used more efficiently, there is a drawback: the worst-case response times of each task and interrupt handler are not immediately obvious. If we don't know what the worst-case behavior is, we cannot sensibly use priority-based scheduling for hard real-time systems. This is where so-called schedulability analysis comes in. Scheduling mathematics called deadline monotonic analysis (DMA) can analyze how tasks scheduled by priorities interact and determine the worst-case response time of each task and interrupt handler.

Before we can go through the basic analysis we need to make some assumptions about how the system behaves and define some basic notation. Following are several assumptions that the analysis makes about how the application and OS behave:

- There is a fixed set of tasks with a fixed and known priority ordering
- Any task can become ready to run at any point in time, but the task cannot become ready again until some minimum time has elapsed
- Each time a task becomes ready to run it will run for only a bounded amount of processor time
- A task may not voluntarily suspend itself during its execution (so waiting for an arbitrary event halfway through executing is not permitted)
- No task deadline can be longer than the task period. This means that the task must finish before it can be due to run again
- Each task must have a unique priority
- A given task is never delayed by a lower priority task. As soon as a task becomes ready to run, the OS switches to running that task. During execution the task is never delayed waiting for a lower priority task to execute. This rules out disabling interrupts and sharing data via semaphores
- The OS performs scheduling and task switching in zero time

The last two assumptions are clearly unrealistic but we'll come back to these shortly and extend the analysis to remove them. The basic analysis can be extended to remove or modify most of the remaining assumptions, and some of these extensions are discussed later. Figure 3 is an annotated time diagram defining the notation used in the analysis.

The term T_i is the so-called "period" of a task *i*. The period is the minimum time between a task being made ready to run and being made ready again. This model supports both sporadic and strictly periodic tasks.

Each time the task *i* is made ready it may execute for up to C_i processor time. This is known as the worst-case execution time of task *i*. Note that this time does not include the time for which other tasks and interrupt handlers use the processor; it is the processor time required only by task *i*.

Before we go on it is worth mentioning a useful technique for finding task execution times. This is to measure execution time for tests that exercise all feasible paths, with the OS providing a logical stopwatch for each task. When the task is made ready, the stopwatch is zeroed. Whenever the task runs, the stopwatch counts. Whenever another task or interrupt handler runs, the stopwatch is stopped. The "high-water mark" of the stopwatch after the tests gives the worst-case execution time.

The worst-case response time of a task *i* is measured from the time the task is made ready to the time the task completes its worst-case execution time C_i. This worst-case response time is denoted R_i. The deadline of a task *i* is denoted D_i and a task will always meet its deadline if $R_i \leq D_i$.

The basic idea of DMA is to find an equation that will calculate R_i. R_i is made up of two times: the time a task takes to execute its own code, and the time it takes for higher priority tasks to execute and finish with the processor. The following equation represents this relationship:

$$R_i = C_i + I_i$$

The term I_i is the pre-emption time from higher priority tasks and interrupt handlers. It's called the "interference." The problem now becomes finding the interference time.

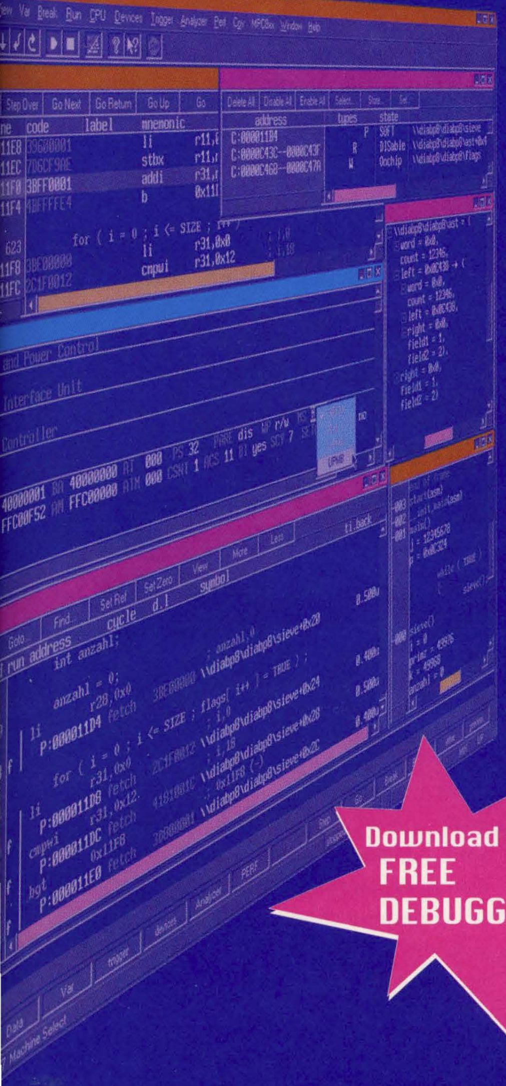
Figure 4 shows a task being pre-empted by a higher priority task. It turns out that the maximum interference from a higher priority task *k* occurs when the lower priority task and task *k* are made ready at the same time. Figure 4 shows how task *k* initially pre-empts the task then pre-empts again when it is made ready for a second time. When task *k* comes back for a third try the lower priority task has already finished, and this time there is no interference.

TRACE32®

IN-CIRCUIT-EMULATORS & DEBUGGERS

ONE IDE SERVES ALL!

TRACE32-PowerView



TRACE32-ICD
the highly cost effective
In-Circuit Debugger

TRACE32-FIRE
the Fully Integrated Risc
Emulator at highest speeds

TEST THE BEST!

- *Professional tools exceed highest demands*
- *The ultimate in technology and speed*
- *Greatest variety of supported CPUs (ARM, C167, H8S, PowerPC, SH2, ST10 and more)*
- *Unique modularity and open concept*
- *Test and compare our comfortable tools by downloading our simulators*

Download
FREE
DEBUGGER!

Lauterbach, Inc.
4, Mount Royal Ave
Marlborough, MA 01752
Fax: 508-303-6813
Phone: 508-303-6812
e-mail: info_us@lauterbach.com

Lauterbach, Inc.
13256 SW. Hillshire Drive
USA-Tigard, OR 97223
Fax: 503-524-2223
Phone: 503-524-2222
e-mail: info_us@lauterbach.com

LAUTERBACH

Free simulators available in the Internet:

http://www.lauterbach.com/download.html

FIGURE 4 Preemption example

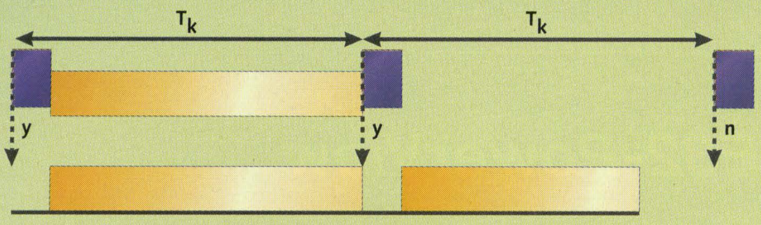


TABLE 1 DMA example

Task	T	C	D
1	250ms	5ms	10ms
2	10ms	2ms	10ms
3	330ms	25ms	50ms
4	1000ms	29ms	1000ms

TABLE 2 Calculation of worst-case Task 3 response time

Step	R^n	I	R^{n+1}
1	0	0	25
2	25	5 + 3 x 2 = 11	36
3	36	5 + 4 x 2 = 13	38
4	38	5 + 4 x 2 = 13	38

The number of times that a given task *k* can pre-empt a task *i* while task *i* is ready is given by:

$$\left\lceil \frac{R_i}{T_k} \right\rceil$$

The symbol $\lceil\ \rceil$ is the ceiling function, and is a round-up function. So, for example:

$$\lceil 1.2 \rceil = 2$$
$$\lceil 2.01 \rceil = 3$$
$$\lceil 2.0 \rceil = 2$$

The total time taken by a higher priority task *k* when it pre-empts and executes is simply given by:

$$\left\lceil \frac{R_i}{T_k} \right\rceil C_k$$

So for the total interference term, we simply add this up for all the higher priority tasks in the system and get the following:

$$I_i = \sum_{\forall k \in hp_i} \left\lceil \frac{R_i}{T_k} \right\rceil C_k$$

The term hp_i is the set of all tasks of higher priority than task *i.* The effects of interrupt handling can be included by ensuring that each interrupt source has a *C* and *T* value and including them in the set hp_i.

Now that we have a term for the interference we can present the basic DMA response time equation:

$$R_i = C_i + \sum_{\forall k \in hp_i} \left\lceil \frac{R_i}{T_k} \right\rceil C_k \qquad (1)$$

Because R_i appears on both the left- and right-hand sides of the equation it would appear that we can only find the worst-case response time if it is already known. Fortunately, this equation can be solved by forming a recurrence relation:

$$R_i^{n+1} = C_i + \sum_{\forall k \in hp_i} \left\lceil \frac{R_i^n}{T_k} \right\rceil C_k$$

An initial value of zero for R_i is sufficient and the sequence will converge to the smallest value of R_i that satisfies Equation 1.

Before we eliminate some of the assumptions we made earlier, it's worth discussing how priorities are chosen in the first place. The term "deadline monotonic" refers to the priority allocation algorithm: assigning priorities monotonically with deadline. Thus the task with the shortest deadline (the smallest value of *D*) is assigned the highest priority. Ties are broken arbitrarily. There are proofs that this is the optimal priority assignment algorithm under the assumption that $T < D$ for all tasks.

Let's try an example. Table 1 describes a set of tasks. The table is in deadline monotonic priority order, with task 1 being the highest priority and task 4 the lowest priority. Notice how we don't care if the tasks are periodic or sporadic: we just need to know the minimum inter-arrival time.

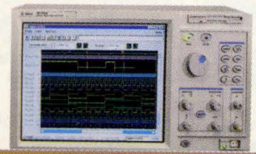
Let's calculate the worst-case response time of task 3. We start with an initial *R* estimate of 0. Table 2 shows the steps in the calculation. The equation converges at 38ms. So the worst-case response time of task 3 is 38ms. This is less than the deadline of 50ms and proves that task 3 will always meet its deadline in all situations.

Priority inversion, priority inheritance, and the priority ceiling protocol

One assumption we have made so far is that a task will never be delayed by a lower priority one. This means that no lower priority task is allowed to disable interrupts or even share a semaphore with a higher priority task. Clearly this assumption is unrealistic and we need to do something about it. The way we deal with this is to allow for so-called "blocking time." Denoted B_i, blocking time is equal to the time for which the execution of lower priority tasks can delay a given task *i.* The equation for the worst-case response time can be updated to take blocking time into account:

$$R_i = B_i + C_i + \sum_{\forall k \in hp_i} \left\lceil \frac{R_i}{T_k} \right\rceil C_k \qquad (2)$$

We need to work out this blocking time. Easier said than done, of course.



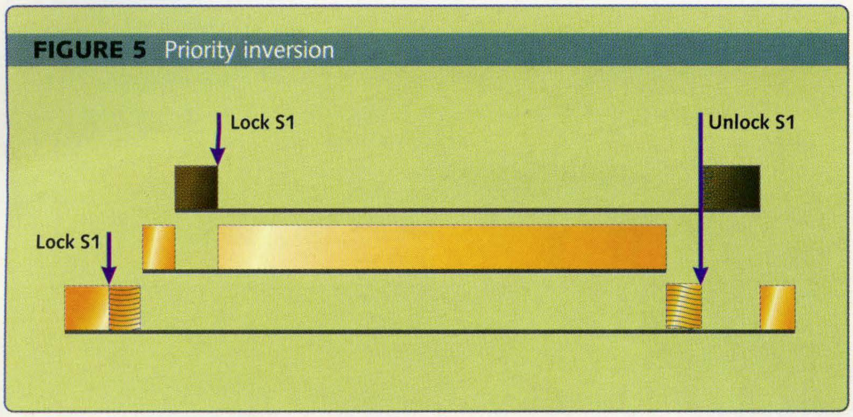


FIGURE 5 Priority inversion

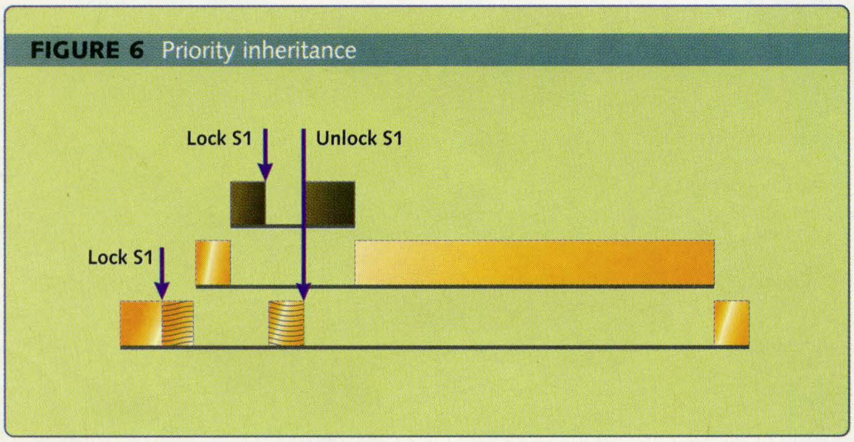


FIGURE 6 Priority inheritance

To see why this can be difficult it's worth taking the time to discuss a common problem known as *priority inversion.* Priority inversion occurs when tasks use regular semaphores to guard access to shared data, and cause the response time of a task to become much longer than normal. This often leads to the system missing deadlines intermittently. Figure 5 shows how the problem can occur.

The diagram is a timeline showing the execution trace of three tasks scheduled by priorities, with the highest priority task at the top and the lowest priority task at the bottom. Two tasks, "L" with a low priority and "H" with a high priority, share a data buffer and need to ensure exclusive access to the data. This is done via a semaphore ("S1"). In Figure 5, L is running and wants to get access to the data. It locks S1 and starts using the data. While accessing the data, a medium priority task ("M") becomes ready. The scheduler suspends L and starts running M. Shortly afterwards, H is made ready, and the scheduler suspends M and starts running H. Then H wants to access the data and tries to lock S1. Unfortunately, S1 is already locked, so H is blocked awaiting the semaphore. The next highest priority task is M, so the scheduler resumes executing M, which runs for some time and eventually finishes, and the scheduler resumes executing L. Then L finishes with the shared data and unlocks S1. Task H is waiting for S1 and so is made ready to run. The scheduler resumes executing H, which then accesses the data, unlocks S1, and finishes.

You can see from Figure 5 how the response time of H is quite long—longer than M in any case—which isn't the right behavior because H should have a shorter response time. H is assigned a high priority because it's urgent and shouldn't end up being delayed for a longer time than M. The problem is called "priority inversion" because task M is delaying task H and, in effect, executing at a higher priority.

One of the problems with priority inversion is that it can be "hidden"—there can be implicit resource contention that's resolved using regular semaphore locks. For example, it is common for an RTOS to provide a threaded C library where calls are protected against concurrent access. For example, the `malloc()` and `free()` calls typically lock an internal semaphore to guard access to their heap management data. When two tasks both make `malloc()` and `free()` calls, they are implicitly sharing data and making semaphore locks. A low priority task might be making a `malloc()` call when a high priority task also makes a `malloc()` call. The high priority task is blocked and medium priority tasks get to run. Priority inversion again! Worse yet, the programmer might not even make `malloc()` and `free()` calls: they might be invoked automatically by a C++ compiler's code for constructors and destructors.

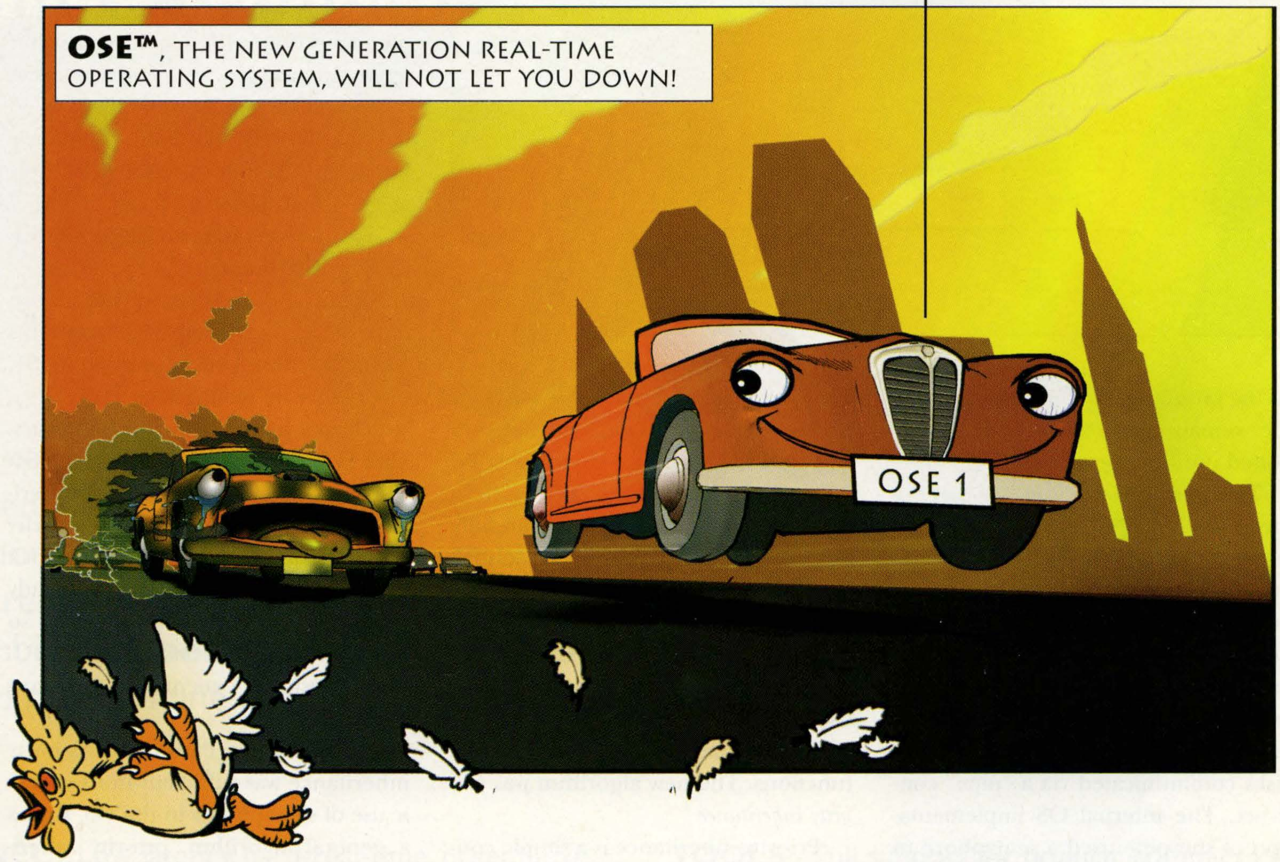
Priority inversion also causes a lot of trouble because it is intermittent: in testing, the scenario shown in Figure 5 might not occur—response time of H will be short. Yet after deployment the tasks might become phased as shown in Figure 5, increasing the response time of H, and causing the system to fail. This is exactly what happened to the "Pathfinder" probe to Mars in July 1997.

The spacecraft was controlled by a single processor on a VME bus that also contained interface cards for the radio, a camera, and an interface to an on-board MIL-STD-1553 bus. The bus connected two parts of the spacecraft together, with the VME system containing interfaces to, among other things, the ASI/MET meteorological science instrument.

In the first few hours of operation on Mars, the spacecraft began experiencing resets. A reset re-initialized all hardware and software and terminat-

OSE

– Conventional RTOS? ...sorry!


OSE™, THE NEW GENERATION REAL-TIME OPERATING SYSTEM, WILL NOT LET YOU DOWN!
OSE 1

OSE THE NEW GENERATION RTOS!

FASTER

Streamlined for extreme reliability and speed, OSE's kernel and TCP/IP stacks blow away the competition. From the RTOS to the tools, OSE sets tomorrow's standards for small size and high performance.

TOUGHER

As the only fault-tolerant RTOS, OSE supports mission-critical real-time systems, allowing complete non-stop recovery from hardware and software failures AND hot swaps – critical for high-availability functionality.

LONGER-LASTING

OSE is heterogeneous, scalable, and distributable, protecting your investment by allowing your application to grow from one CPU to hundreds.

MORE POWERFUL

OSE's kernel offers automatic supervision, dynamic reconfiguration and integrated error handling, letting you focus on your core competency: designing applications.

SIMPLER

OSE's powerful ultra-efficient message-based architecture lets you write nearly every bit of application code using only eight system calls.

SAFER

OSE is the world's only RTOS that is safety certified to the demanding specifications of IEC 61508 . OSE is also being certified according to the stringent DO 178-B.

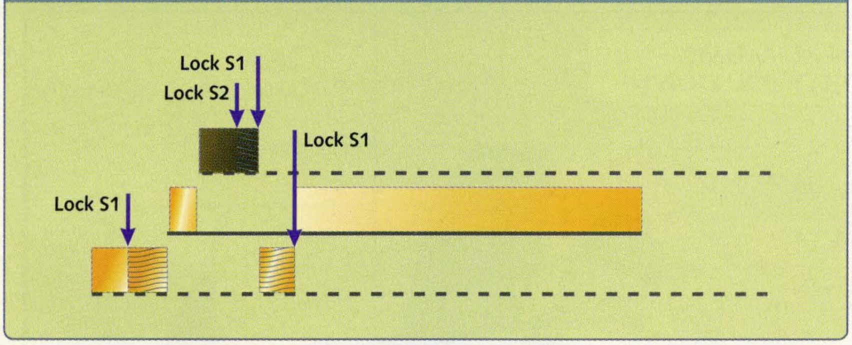
PROVEN

Millions of products worldwide are already taking advantage of OSE , including the top brands in telecommunications and process control. OSE is the RTOS of the future.

www.enea.com

ENEA OSE SYSTEMS INC
5949 SHERRY LANE , SUITE 625, DALLAS TX 75225. PHONE: 214-346-9339. FAX: 214-346-9344. EMAIL: info@enea.com

FIGURE 7 Priority inheritance and deadlock



ed the ground command activities and the remaining activities were postponed until the next day.

The resets were due to priority inversion. The JPL engineers used a conventional RTOS to implement the bus control functions in the main processor. A task with a 125ms period, but a short deadline—and so a high priority—was driving the on-board bus and handing data from the instruments to a low priority task. The two tasks communicated via a "pipe" construct. The internal OS implementation of the pipe used a semaphore to guard internal OS data.

On Earth extensive testing was done but the problem only manifested itself when the ASI/MET data was beiňg collected at the same time as a high load due to medium priority tasks. Although one reset was seen in testing, the problem was not repeatable and could not be isolated. Because the problem was rare the system was deemed sufficiently reliable (the space business is inherently risky and there's no point engineering a limited-life space vehicle to a much higher reliability than the launch technology).

On Mars the task execution patterns were different and priority inversion occurred frequently. Medium priority tasks stopped the low priority instrument task from running, which stopped the high priority bus task from running. When the bus task missed its deadline the system assumed a fault and resets itself. However, since the fault was a design fault—using conventional semaphores in a real-time system—restarting the system just triggered the fault again.

Space systems are usually carefully designed to let Earth-based control gain access and upload new software, so the JPL engineers were able to patch over the problem by uploading new software that instructed the OS to use a different algorithm to lock the internal semaphores used in the pipe functions. The new algorithm was *priority inheritance.*

Priority inheritance is a simple concept: a lower priority task L delaying a higher priority task H inherits the priority of H while it's causing a delay. Once L unlocks the semaphore, the priority of L falls back. Figure 6 shows this.

Task L locks S1 as before and is preempted by M and then H. When H tries to lock the semaphore, the priority of L is raised to high and so L resumes execution. When L releases the semaphore, its priority is restored to low. Now that the semaphore is free, task H resumes execution and finishes. Then M is resumed, and finally L completes. The high priority task has a shorter response time and in the case of the Pathfinder probe, this was sufficient to obtain correct operation. The rest of the Pathfinder mission was a success (the mission returned 2.6 billion bits of information, including more than 16,000 images).

Unfortunately, priority inheritance doesn't solve the problem of unpredictable response times. We wanted to be able to work out B_i, but priority inheritance doesn't help. For example, it's possible for a system to suffer from "circular blocking," where task A is blocked by task B, which is blocked by task A. This is more familiarly known as "deadlock" and effectively gives infinite blocking times. Figure 7 shows how two tasks can deadlock with priority inheritance.

In Figure 7, tasks L and H share two data buffers, guarded by semaphores S1 and S2. Task L locks semaphore S1, but before it manages to lock S2, a switch to a higher priority task occurs. Task H locks S2, then tries to lock S1. Through the rules of priority inheritance, task L inherits a high priority and resumes execution. L tries to lock S2, but unfortunately S2 is already locked. Neither H nor L can run, so task M resumes execution. H and L never resume execution; they are deadlocked.

For the Mars Pathfinder, priority inheritance was sufficient in its specific use of semaphores in the OS. But as a general algorithm, priority inheritance is not effective. In general it's not possible to show a system is free of deadlock and it's impossible to determine B_i for each task i in the system.

Fortunately, there is a solution to this problem: the Priority Ceiling Protocol. This protocol provides a key property, in which a high priority task is blocked once at most by all lower priority tasks. This allows the blocking times to be calculated, and hence for DMA to be used to verify timing performance. Although the original algorithm as specified in 1987 is more complex than priority inheritance, there is a low-cost variant called Instant Inheritance that has identical worst-case behavior (and so the same blocking times apply) but can be implemented very efficiently.

The Priority Ceiling Protocol introduces the notion of "ceilings." Each semaphore has a ceiling priority: this is the priority of the highest priority task that can lock the semaphore. A

FIGURE 8 Instant inheritance variant of the priority ceiling protocol

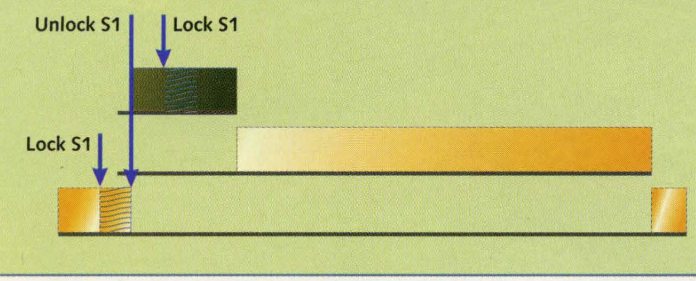
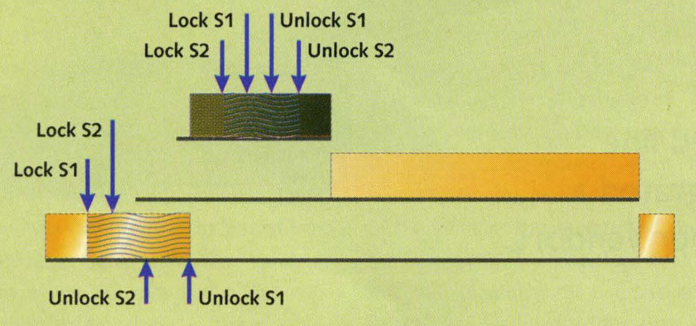


FIGURE 9 Priority ceiling protocol avoids deadlock



task can hold several semaphores at once, but only if they are locked in a nested pattern (for example, lock S1 .. lock S2 .. unlock S2 .. unlock S1). The Instant Inheritance algorithm is very simple: when a semaphore is locked, the locking task raises its priority to the ceiling priority of the semaphore. When the semaphore is unlocked the task's priority is restored. Figure 8 shows this.

The scenario in Figure 8 is the same as in Figure 3. Task L locks semaphore S1. Because both L and H share S1, the ceiling priority of S1 is high. When the lock call takes place the priority of L is raised to high. When M is made ready to run, no switch occurs because task L is running with a higher priority. Similarly, no switch occurs when H is made ready to run. Then when L unlocks S1 its priority is restored to low, and a switch to H occurs. After H has finished, task M executes. Finally, task L is resumed and completes. You can see that the response time of H is short, which is just what's wanted.

Figure 8 shows why the Immediate Inheritance variant is so easy to implement. The "lock" call always succeeds (see how when L and H lock S1, neither is blocked). The reason the call always succeeds is that no other task can have the semaphore locked, because such a task would have the ceiling priority of the semaphore and would be running already, and the current task would not be running. So the lock call always succeeds. This means that implementing semaphores merely requires a "change the current priority" operation. There is no need to queue the task in a per-semaphore queue or even test to see if the semaphore is locked.

Most importantly, the blocking times of tasks can be worked out. A given task *i* is blocked at most once by a lower priority task. In fact, the blocking time of a task *i* is the longest time any lower priority task *k* holds a semaphore with ceiling priority greater than or equal to the priority of task *i*. Because the blocking times are bounded there is no chance of deadlock. Figure 9 shows this. The scenario is the same as in Figure 7 but this time deadlock is avoided.

In Figure 9 the ceiling priority of semaphores S1 and S2 is high because both L and H lock these semaphores. As soon as L locks S1, the priority of L becomes high. This prevents both H and M from running. Locking S2 has no effect because L is already running at high priority. When L unlocks S1, its priority falls back to low, and task H gets to run, lock both semaphores, and finish.

We can more formally define the blocking time for a given task *i* by defining the following terms:

- Let lp_i be the set of tasks with priorities lower than task *i*
- Let $locks_{k,i}$ be the set of semaphores locked by a task *k*, where the semaphores have a ceiling priority higher than the priority of task *i*
- Let $t_{k,s}$ be the time for which a task *k* holds a semaphore *s*
- The blocking time B_i is defined as follows:

$$B_i = \max_{\substack{\forall k\in lp_i \\ \forall s\in locks_{k,j}}} \left(t_{k,s}\right) \qquad (3)$$

Figure 10 shows how this works for task M in the simple system shown earlier in Figure 5.

The observant reader might notice that the Immediate Inheritance variation of the Priority Ceiling Protocol is much like the approach of raising the processor interrupt priority level to hold out interrupt handlers of a certain priority so that data shared with those handlers isn't corrupted. In effect the protocol has re-invented, albeit in a more systematic way, what was first done in the 1960s. Because there is a contiguous range of priorities, from software tasks to hardware

WHAT DOES

DEVICE SERVER™

TECHNOLOGY

MEAN TO YOU?

THE MOST POWERFUL WAY TO LEVERAGE THE INTERNET.

INTERNET CONNECTIVITY. IT'S NO LONGER A QUESTION OF "WHY." TODAY, YOU HAVE TO FIGURE OUT "HOW." HOW CAN YOU GAIN ACCESS TO VITAL INFORMATION THAT IS PRESENTLY TRAPPED IN UNCONNECTED DEVICES? HOW CAN YOU LEVERAGE THE INTERNET TO SOLVE YOUR BUSINESS PROBLEMS?

PUT THE INTERNET IN...

THE ANSWER IS SIMPLE – PUT THE INTERNET IN YOUR DEVICES. GIVE DEVICES THE POWER AND FREEDOM TO SHARE INFORMATION DIRECTLY OVER THE INTERNET WITHOUT THE NEED FOR BULKY, EXPENSIVE PCs. PLUS, YOU'LL ALSO HAVE THE ABILITY TO ACCESS CRITICAL INFORMATION, MANAGE DEVICES IN REAL-TIME, AND CREATE INTERNET-ENABLED APPLICATIONS THAT YOU NEVER IMAGINED POSSIBLE. LANTRONIX DEVICE SERVER™ TECHNOLOGY MAKES THIS POSSIBLE FOR MILLIONS OF PEOPLE EACH DAY. OUR DEVICE SERVER TECHNOLOGY HAS BEEN SUCCESSFULLY DEPLOYED IN MILLIONS OF DEVICES ENABLING THEM TO TALK DIRECTLY OVER THE INTERNET, SAVING ORGANIZATIONS MILLIONS OF DOLLARS. DEVICE SERVERS ARE OPENING UP HUGE NEW BUSINESS OPPORTUNITIES FOR SYSTEM INTEGRATORS, ALLOWING THEM TO CREATE AMAZING INTERNET-ENABLED APPLICATIONS FOR THEIR CUSTOMERS. AND MANUFACTURERS ARE PUTTING THE INTERNET INTO THEIR DEVICES IN WEEKS INSTEAD OF YEARS. THE POST PC ERA HAS ARRIVED. ARE YOU READY?

DEVICE SERVER TECHNOLOGY BENEFITS

- Embed the Internet in any device
- Give devices the power and freedom to share critical information
- Fast time to market—weeks instead of years
- Save thousands of dollars in deployment and connectivity costs
- Create Internet-enabled applications that were never before possible
- Gain a new competitive edge
- Develop new business opportunities and generate new revenue streams
- Access, manage and control devices from anywhere in real-time

WITH DEVICE SERVER™ TECHNOLOGY, YOU CAN INTERNET-ENABLE ANY DEVICE.

past

Bring the benefits of Internet connectivity to your installed devices, equipment and software applications with Device Servers, giving them new life and immediate access to the network. Device Servers transparently handle all communications, letting you use existing applications without any modification.

future

Device Server Technology is the perfect enabling platform for manufacturers facing increasing demand for the Internet in their future products. By embedding board-level Device Servers or licensing Device Server Technology, OEMs can focus on core competencies and bring amazing new products to market faster than ever before.

present

Device Servers can be bundled with non-networked devices you are currently deploying for immediate Internet connectivity. Device manufacturers can also embed board-level Device Servers into current products to Internet-enable them in a flash, hardware and software integration.

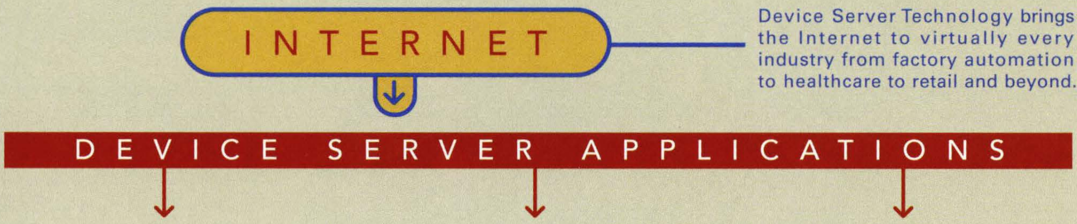
Device Server Technology Defined

Lantronix Device Server Technology is a powerful, turn-key solution that lets nearly any device – past, present or future – communicate directly over the Internet. We've refined and optimized Device Server Technology over the past decade to provide powerful, flexible, connectivity and computing power at the device level. Unlike bulky expensive traditional PCs, Device Servers are the perfect computing platform for the Post PC Era.

Device Servers range from rack-mount multiport units down to single port servers that are smaller than a deck of playing cards. Board-level Device Servers come as small as a book of matches, allowing you to embed the Internet into nearly any device, regardless of size. Device Servers are easy to manage using any standard web browser, thanks to a built-in HTTP server and Java management program. And with our Software Developer's Kit, you can even write your own customized web-enabled software for your Device Servers.

The driving force behind Device Server Technology is highly optimized applications firmware that has been refined, and proven in the field to provide access, management and control of nearly any device over the Internet. Critical features like modem emulation, hardware flow control, specialized timing and custom protocol support are all there to ensure your applications work the first time and every time. Unlike other communications technologies, Lantronix Device Server Technology is based on widely accepted industry standards to ensure compatibility today and in the future.

THE NEW COMPETITIVE EDGE



Device Server Technology brings the Internet to virtually every industry from factory automation to healthcare to retail and beyond.

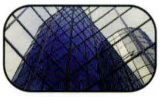
HOSPITAL INFORMATION ACCESS

A leading manufacturer of handheld blood analyzers uses Device Servers to quickly transmit vital patient data from the bedside to the hospital information system without using PCs.

INTERNET ENABLED SECURITY SYSTEMS

Access card readers and door locks can now be controlled and reprogrammed over the building network, instead of using local control panels.

REMOTE BUILDING HVAC CONTROL



Building administrators can now access and control up to 8,000 thermostats from a single PC to keep tabs on heating and cooling in massive apartment complexes.

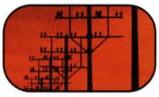
COST-EFFECTIVE RETAIL POS CONNECTIVITY

A major retailer dramatically reduced the operating costs of event ticket sales kiosks in their stores by using Device Servers to eliminate expensive dial-up lines.

ENTERPRISE IT MANAGEMENT

Device Servers allow multinational organizations to manage reserved console functions for mission-critical networking equipment in corporate data centers around the world from a single workstation.

INDUSTRIAL CONTROL AND REMOTE MANAGEMENT



A leading energy company is using Wireless Device Server Technology to remotely manage their power grid in real time through a web browser application.

These are just a few of the applications that are benefitting from the Internet revolution thanks to Device Server Technology. No matter what industry you are in, Device Servers give you the freedom to create amazing new Internet-enabled applications and products today. The possibilities are limited only by your imagination.

For more information on Device Server Technology and how it can make the Post PC era a reality for you, please visit our website at **www.lantronix.com** or call us toll free at **877-462-9682.**

15353 Barranca Parkway
Irvine, California 92618 USA
949-453-3990
sales@lantronix.com

LANTRONIX

www.lantronix.com
Europe: +31-76-565-8176
Asia: +65-447-4222

interrupt priorities, the disabling of interrupts using the mechanism described can be modeled by DMA. Disabling interrupts to a particular level is just like locking a semaphore with a ceiling equal to that level.

FIGURE 10 An example of blocking time

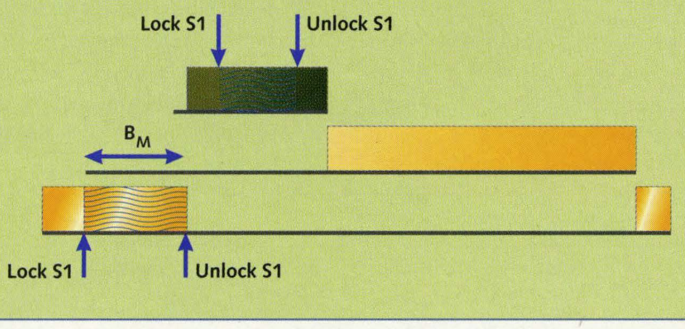
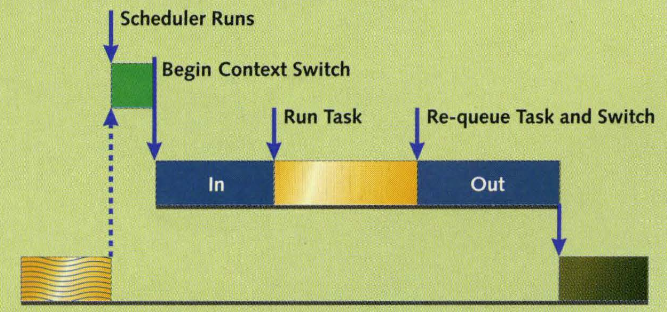


FIGURE 11 Switching costs and the execution of a task



OS overheads: scheduling and switching

The second major assumption we made earlier in DMA was that the OS performed scheduling and task switching in zero time. Clearly this is not feasible, and we must do something about it.

We first look at bounding the costs of the scheduler. The scheduler is the part of the OS that decides when to apply delays and timeouts, so that tasks are ready when due. A common implementation is a so-called "tick scheduler." A periodic timer interrupt calls the OS scheduler that processes a time-ordered delay queue. The scheduler takes due tasks from the time queue and inserts them into a priority-ordered "ready" queue. At the end of the tick the OS calls the dispatcher if a task switch is necessary. The scheduling approach requires that tasks handled by the tick scheduler have periods that are integer multiples of the tick period and at least as large. So if the tick period is 10ms then a task could have a period of 10ms, 20ms, 30ms and so on, but not 25ms. Similarly, none of these tasks can have a period shorter than 10ms.

The overheads of the scheduler would normally be modeled by including the timer interrupt as a task (as we would for all other interrupt sources). The worst-case execution time of the interrupt handler is the interrupt overheads plus the longest execution time of the scheduler. This latter time may not be not bounded easily because typically, the queue processing depends on how many tasks are in the system: the more tasks, the longer the scheduler takes to run.

The following equation is a bound on the scheduler overheads in an interval of duration *t*:

$$\left\lceil \frac{t}{T_{tick}} \right\rceil C_{tick}$$

where T_{tick} is the tick period and C_{tick} is the worst-case execution time of the tick scheduler.

Although this is sufficient, the overheads bound turns out to be quite pessimistic. This is because it assumes that for each tick all tasks are in the delay queue and that the whole queue is processed in the same tick. Although this can happen, it's rare. We can get more realistic bounds by realizing that the tick scheduler processes any given task *i* at most once every T_i. In effect, there is a pseudo-task for each task *i*, with a worst-case execution time equal to the per-task cost of the tick scheduler and a period equal to T_i.

The following equation is a more accurate bound on the scheduler overheads in an interval of duration *t*:

$$\left\lceil \frac{t}{T_{tick}} \right\rceil C_{base} + \sum_{\forall k \in ticktasks} \left\lceil \frac{t}{T_k} \right\rceil C_{task}$$

where C_{base} is the execution time of the tick scheduler when the tick processes zero tasks, C_{task} is the additional per-task cost of the tick, and *ticktasks* is the set of all tasks that are scheduled by the tick scheduler. Note that C_{task} is often not a simple constant but rather a function of how many tasks are in the system.

From this we can derive an equation that bounds the time a given task *i* is delayed by the execution of the scheduler:

$$S_i = \left\lceil \frac{R_i}{T_{tick}} \right\rceil C_{base} + \sum_{\forall k \in ticktasks} \left\lceil \frac{R_i}{T_k} \right\rceil C_{tasks} \qquad (4)$$

We can now update Equation 2 to include the scheduling costs:

$$R_i = S_i + B_i + C_i + \sum_{\forall k \in hp(i)} \left\lceil \frac{R_i}{T_k} \right\rceil C_k \qquad (5)$$

Now we can look at the task switching costs. The OS dispatcher performs task switching and is called whenever a task switch is necessary. For example, the scheduler may call the dispatcher if it makes a new task ready that is of higher priority than the current task. OS calls themselves may also call the dispatcher. For example, a semaphore "unlock" call may necessitate a task switch, triggering a call to the dispatcher. Whenever the dispatcher is called, it does a simple job: it saves the context of the current task (that is, processor registers), restores the context for the new task, and resumes the execution of this new task. Later, when the new task finishes, the dispatcher is again called and saves the context, places the task in a non-ready state, chooses which task to run next, and then restores the context of the next task.

The "switch-in" cost to start a new task usually has a simple bound, but the "switch-out" cost is often more complex, because placing the task in a non-ready state can entail searching a time-ordered queue of timeouts to insert the task in the right place. So the "switch-out" cost is usually dependent on the number of tasks in the system.

The switching costs of the OS can be modeled by adding the "switch-in" and "switch-out" costs to the worst-case execution time of each task. Figure 11 shows this. Note that C_{out} is often not a simple constant and is typically a function of how many tasks are in the system.

The total switching costs for a single switch are, of course:

$$C_{sw} = C_{in} + C_{out} \qquad (6)$$

Using this definition, we can update Equation 5:

$$R_i = S_i + B_i + C_{sw} + C_i + \sum_{\forall k \in hp_i} \left\lceil \frac{R_i}{T_k} \right\rceil (C_{sw} + C_k) \qquad (7)$$

Where:

- C_i is the worst-case execution time of a given task i
- T_k is the minimum time between a given task k being made ready and being made ready again
- C_{sw} is costs of switching to and back from a pre-empting task and is defined by Equation 6
- B_i is the blocking time of a given task i and defined by Equation 3
- S_i is the scheduler overheads for a

WHAT'S LURKING IN YOUR CODE?
WE CAN TELL YOU. FAST.
EARLY.
WITHOUT TESTING.

given task i and is defined by Equation 4

- hp_i is the set of tasks of higher priority than a given task i

Discussion

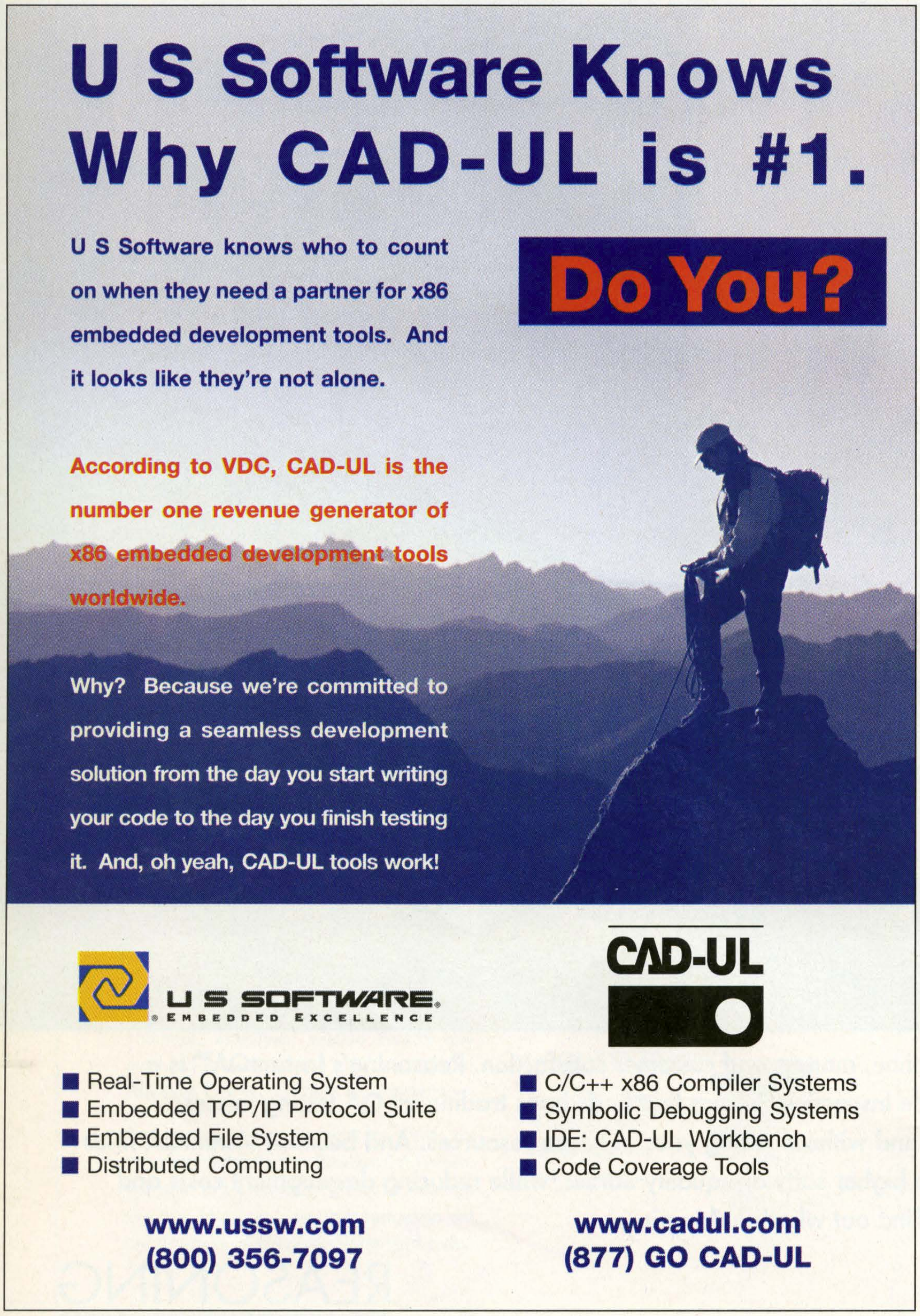
With Equation 7 and the right OS and application behavior, we calculate the worst-case response time for every task in a system. This is useful because it means we can verify the timing behavior of complex real-time systems before deployment. But the analysis can be applied to more than just processor scheduling.

Controller area network (CAN) is an embedded networking bus that arbitrates between messages on the bus by using priorities. Each message is tagged with a unique priority which serves to identify the message. Because the bus is scheduled by priorities the basic DMA is a natural fit and has been developed to calculate worst-case latencies for messages sent on a CAN bus. The Volvo Car Corp. used this analysis as the foundation of the networking system for the Volvo S80 sedan. The analysis was used as part of an optimizing toolchain to configure the network so that the lowest bus speeds could be used (minimizing wiring costs), yet with all timing requirements met. This vehicle has two CAN buses, and the complete timing behavior of the network was verified using the analysis before the vehicle went into series production.

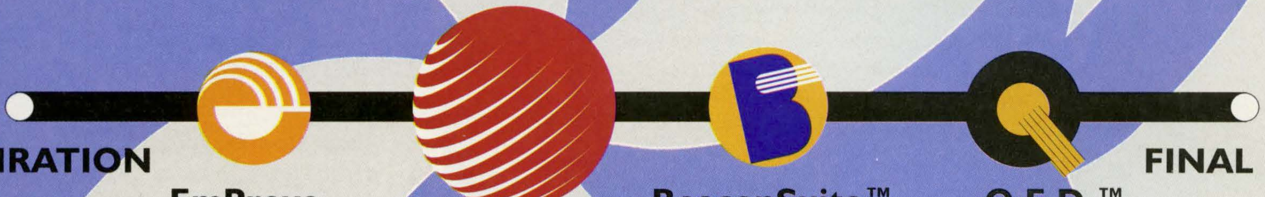
DMA has also been applied to the problem of scheduling a hard disk drive to fetch and store data for concurrent multimedia streams. Each data stream requires a bounded number of disk blocks every so often. This is analogous to each task requiring a bounded computation time every period. The time taken to move the disk head between the disk areas for different data streams is analogous to task switch times. DMA for disk drive scheduling allows a playback video server to apply the analysis each time an "open" request is made and to accept the request only when the analysis can guarantee that the data will be fetched in time.

There have been a number of extensions to the basic analysis that remove some of the assumptions listed earlier or else improve the flexibility of the analysis. Examples of these extensions are:

- $D > T$. Allowing the deadline (and hence response time) to be longer than the period turns out to be particularly useful for analyzing wide-area networks carrying multimedia data. For example, a source for a 10Hz video stream may send frame



The RTOS for Internet Appliances, Systems-On-A-Chip, and all other Embedded Systems



INSPIRATION

FINAL CODE

EmProve
Virtual Application Development and Prototyping (VAD-P™) environment

RTXC™
Internet Connected Multitasking Real-Time OS

BeaconSuite™
- Compilers
- Linkers & Locators
- Debuggers
- Simulators

Q.E.D.™
- JTAG Debugger
- 186 /386 ICE

The shortest distance between two points includes the ground we've already covered for you.

We have covered that ground with the most useful, flexible, scalable RTOS you can imagine for DSP, 32-, 16-, and 8-bit embedded applications.

It's what we do for a living - building it right so that you don't have to. You *could* write your own d.o.s. (dumb old scheduler), and *then* try your hand at integrating some file, graphics, and network components.

But why?

As a proven OS, **RTXC** has a pre-emptive scheduled multitasking real-time kernel with an extensive set of services and subsystems built in to assure top performance.

For example, **RTXCnet** includes a full array of networking components - from TCP/IP or PPP to web browser/server. All RTXC components are integrated to kernel resources and architected to fully utilize pre-emptive multitasking. We blow the competition's polling-based architectures away - and so will you! Just pick the networking components you need and RTXCnet arbitrates synchronization and interactions between your application, network stack and kernel.

RTXCfile32, our MS-DOS compatible file management subsystem can be used on any embedded system that uses RTXC - without regard to the target processor. We have drivers for CD ROM, floppy disks, hard disks, PCMCIA memory cards, and FLASH or RAM disks. Our drivers can be tailored to meet your most demanding file-oriented storage device needs.

Our **RTXCgraphics** subsystem endows your development effort with an existing portable graphic user interface library. Video device support is superlative, and fully supports the Rapid Application Development Standard.

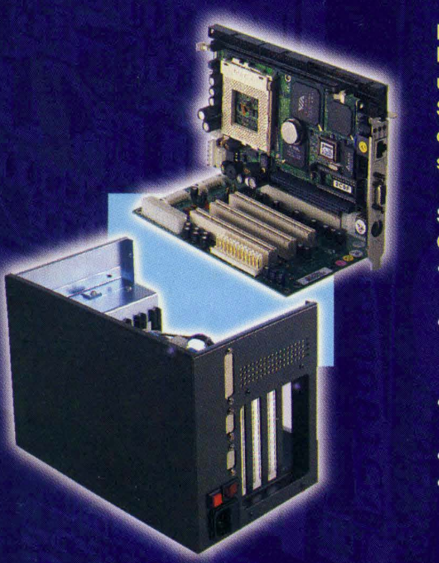
Supported Processors

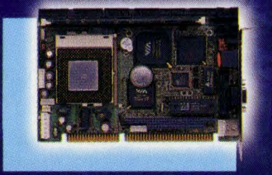
AMD: x86xx Family, ELAN 3/4/520C, K6
ARM: 6, 7/7T, 710, 9T, StrongArm
Hitachi: H8/300 Family
IBM: PPC 4/6
Infineon: 80C16x Family
Intel: 80x86/386/486 Real Mode,
80x86/386/486/Pentium Family Protected Mode
Mitsubishi: M16C
Motorola: DSP, StarCore, M.CORE, MPC500,
ColdFire 52xx/53xx, 68K Family, CPU 32 Family,
68HC16 Family, 68HC12 Family,
and 68HC11 Family
Philips: 80C51XA
ST Microelectronic: ST10
Texas Instruments: 320C3x/5x/20x/54x

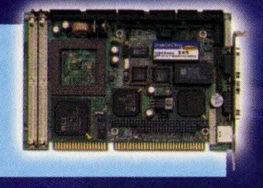
www.embeddedpower.com

Embedded Power
CORPORATION

U.S. 281-561-9990
email: info@embeddedpower.com
Europe: +44 (0) 1256 474448
email: eurosales@embeddedpower.com







data every 100ms, yet the worst-case latency through the network for a single video frame might be two seconds. The analysis has been extended to take into account how previous invocations of a task can impact later ones

- Shared priorities. It's quite common for an implementation to limit the total number of priority levels for efficiency reasons. The analysis has been extended to take account of the behavior of the system when two tasks can have the same priority
- Task phasing. The basic analysis assumes that there is no fixed phase relationship between tasks. But if one task is always made ready some fixed time after another then the basic analysis becomes pessimistic. The analysis has been extended to take account of phase relationships between given sets of tasks

Of course, underlying all this analysis is the assumption that the system is predictable. As we have seen already, the OS must provide the right kind of semaphore locking to the user (the Priority Ceiling Protocol) and also provide a characterization of the scheduling and switching overheads of the OS. The OS must also be carefully designed to avoid internal priority inversion. **esp**

Ken Tindell is a professor of embedded systems at Jonkoping University in Sweden and the chief technology officer of Realogy, a company that provides an RTOS and analysis tools based on extended DMA. He received his PhD in real-time systems from the University of York in England. Since 1995, he has worked in industry, applying real-time techniques to communication and computing. He can be reached at ktindell@realogy.com.

Further reading

You can find out more about hard real-time systems and tools that support DMA at *www.realogy.com.*

BIG FEATURES
SMALL FOOTPRINT

U S Software products are compatible with:

- x86
- 386/486PM
- 680x0/683xx,
- DragonBall EZ
- PowerPC
- ARM
- StrongARM
- SH 1, 2, 3, 4
- SPARCLite
- i960
- NEC V85x
- M*Core
- ColdFire
- MIPS
- C166
- CR16
- 8096/196
- 68HC11
- 68HC16
- 80251
- Z80/180

See our website for additiona microprocessors we support.

With thousands of embedded applications and 25 years in the business, U S Software continues to build cutting-edge, yet compact development tools.

Our software has a history of acclaimed performance and features:

- Compact – Even with industry-leading innovations our small footprint saves you valuable space.
- Fast – Optimized design ensures quick, efficient performance.
- Customizable – Source code is included to give you complete control.*

In the beginning there was silicon. Then, there was U S Software. And throughout, our engineers have crafted software and provided technical support designed to ensure your success.

Call us today at **800-356-7097** and entrust your next embedded project to the engineers and products at U S Software.

- **USNET®** Embedded TCP/IP protocols, Internet Access Package, SNMP and Embedded Webserver.
- **SUPERTASK!™** Multitasking RTOS development suite. ITRON 2.x/3.x API available.
- **USFILES®** Windows/DOS compatible file system with CompactFlash and CD-ROM support.
- **NETPEER™** Distributed networking platform with comprehensive desktop interface including GUI, database, multimedia and more.
- **SUPERPEG™** Embedded GUI for SuperTask!®
- **IRPRO™** IrDA protocol stack with SIR and FIR

800-356-7097

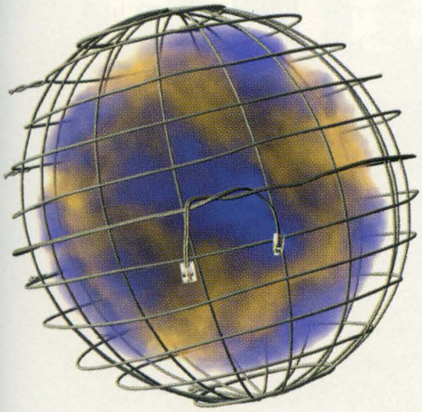
WWW.USSW.COM

*Netpeer™ is delivered as a library.

U S SOFTWARE®
EMBEDDED EXCELLENCE

7175 NW Evergreen Pkwy. Suite 100 Hillsboro, OR 97124
TEL: 503-844-6614 • FAX: 503-844-6480 • EMAIL: info@ussw.com • www.ussw.com

Michael Barr



MAC Daddy

Internet Appliance Design

in this issue

Any book on TCP/IP will tell you that each network-connected device has a globally unique hardware address. But where, exactly, do these addresses come from? And why is it that the authors of books on TCP/IP never bother to answer this or any other practical implementation question?

Plenty of good books are available about TCP/IP and I have no desire to rehash the material covered there. (See the Resources sidebar near the end of this month's column for an annotated list of my favorites.) Rather, I hope to supplement the discussion found in those books with the practical information embedded developers need most. I begin by tackling some of the oft-ignored practicalities in and around embedded networking.

Turtles all the way down

Before we go on, I want to make sure you have a solid picture in your head. That picture is the one in Figure 1, which shows how the TCP/UDP/IP stack fits in between your application code and the network interface. What I want you to understand from this figure is the black box nature of the stack. From the application programmer's point of view, a TCP/UDP/IP stack is nothing but an API to be called. In other words, data is sent and received over the network via a set of well-defined function calls. Typically, the so-called "sockets" API is used.

Figure 1 also shows the black box nature of the network interface. In an ideal world, the device driver interface would remove the stack developer from any knowledge of the actual interface type. In such a case, the same stack could communicate unchanged over an Ethernet or Token Ring network or via a serial or modem link, through the same set of device driver calls. In practice, however, this is difficult to achieve.

You'll see as we go along that the implementation of the lowest layers of the TCP/UDP/IP stack is inextricably linked with the device driver and particulars of the hardware below. This is one of those practical realities rarely dealt with in books.

The convenient end

Another practicality that must be dealt with in a TCP/UDP/IP stack implementation is the issue of endianness. By definition, any computer system must be either:

- *big endian*—it stores the most significant byte of any multi-byte data field at the lowest memory address, which is also the address of the entire field
- *little endian*—it stores the least significant byte of any multi-byte data field at the lowest memory address, which is also the address of the entire field

The 68k and PowerPC are big endian processors, while the x86 and Alpha are little endian. The endianness difference can cause problems if a computer is, unknowingly, trying to read binary data written in the opposite format from a shared memory location or a file. To see what I mean,

But what if you want your stack to be portable so it will run on processors of both types? Well, then, you'll have to decide whether to do this reordering or not, either on the fly or at compile time.

FIGURE 1 The big picture

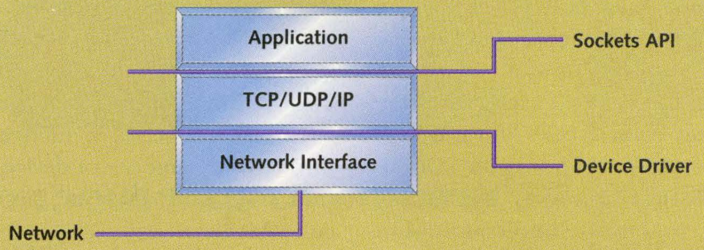


FIGURE 2 Little endian memory dump

```
char  c1 = 1;
char  c2 = 2;
short s  = 255; //0x00FF
long  l  = 0x44332211;

Offset:  Memory Dump
0x0000: 01  02  FF  00
0x0004: 11  22  33  44
```

take a look at an x86 memory dump with a 16- or 32-bit integer stored inside. (If you don't have such a memory dump handy, take a look at Figure 2.) An x86 processor stores data in memory with its least significant byte first. However, your mind tends to expect the data to read from the most significant byte to the least.[1]

Unfortunately, it's not just processors that have this property. Communication protocols, file formats, and all sorts of other computer-related specifications require an endianness of their own. As it turns out, all of the protocol layers in the TCP/IP suite are defined to be big endian. In other words, any 16- or 32-bit value within the various layer headers (for example, an IP address, a packet length, or a checksum) must be sent and received with its most significant byte first.[2]

This, of course, leads to an annoying little implementation detail. If your TCP/UDP/IP stack will run on a little endian processor, you'll have to reorder—at run time—the bytes of every multi-byte data field within the various layer's headers. If your stack will run on a big endian processor, then you have nothing to worry about. But what if you want your stack to be portable so it will run on processors of both types? Well, then, you'll have to decide whether to do this reordering or not, either on the fly or at compile time.

The typical solution to the endianness problem is to define a set of four preprocessor macros: `htons()`, `htonl()`, `ntohs()`, and `ntohl()`, as shown in Listing 1. These macros make the following conversions:

- `htons()`—reorder the bytes of a 16-bit value from processor order to network order. The macro name can be read "host to network short"
- `htonl()`—reorder the bytes of a 32-bit value from processor order to network order. The macro name can be read "host to network long"
- `ntohs()`—reorder the bytes of a 16-bit value from network order to processor order. The macro name can be read "network to host short"
- `ntohl()`—reorder the bytes of a 32-bit value from network order to processor order. The macro name can be read "network to host long"

If the processor on which the TCP/UDP/IP stack is to be run is itself big endian, each of the four macros will be defined to do nothing and there will be no run-time performance impact. If, however, the processor is little endian, the macros will reorder the bytes appropriately. As we get into the details of the IP and UDP layer implementations, you'll see that these macros are routinely called when building and parsing network packets. Only when we see how frequently they are called will we be able to assess the true cost of implementing a big endian protocol stack on a little endian processor.

Hidden assumptions

If you read Christopher Leidigh's recent article ("Configuring TCP/IP Hosts," April 2000, p. 71), you're probably aware that simply adding a protocol stack and network interface to your software doesn't make your system TCP/IP-ready. Among other things, each system requires an IP address to communicate over the network. Leidigh's article did a great job of showing us how to bootstrap a system, determining the IP address in the process. But did you notice the way he assumed (like everyone else) that your system already had a unique hardware address?

"Upon receipt [of the broadcast packet], a RARP server will look up the client's hardware address in its tables [and return the IP address allocated to the client]" (p. 75).

Where does the "client's hardware

Make Your Embedded Platform Soar! with Treck Inc.

Real-Time Internet Protocols for Embedded Systems

We know that schedules are always tight, and that you do not have time to "port" Internet protocols into your embedded environment (let alone write them from scratch). That does not stop marketing people inside your company from asking for Internet functionality in your new product line that is rolling out in the next few weeks. Now you have someone to turn to.

High quality, high performance Internet protocols are what makes us the #1 choice for designers of embedded systems who need Internet connectivity. We design and implement our products specifically for embedded systems. That is why we are three times faster than our fastest competitor. Of course our Internet protocols do not require any porting to your specific platform. This means that you will never need to change a single line of our ANSI "C" code. You do not need an RTOS to use our Internet protocols, but if you have one, we will certainly work with it.

We are very committed to supporting you in your development effort. We know that you do not have time to wait for someone to call days later to get a support answer. That is why we provide the "finest" support in the industry. You can always count on Treck Inc. to help you every step of the way.

Our base product (Treck Real-Time TCP/IP), includes Sockets, TCP, UDP, ICMP, IP, Ethernet, ARP, SLIP and Ping. Treck also provides support for PPP, DHCP, BootP, FTP, and Telnet. All of our Internet products arrive at your door in source code form and are always royalty free.

Call us today to see how Treck Inc. can help you!

Treck Inc.

(800)340-6648 or (513)688-0553

or visit our web page at www.treck.com

Every company that wants to produce Ethernet-capable products must first register with the IEEE. You start by filling out some paperwork and sending them a check for $1,250.

LISTING 1 Byte reordering macros.

```
#if defined(BIG_ENDIAN)

#define htons(A)  (A)
#define htonl(A)  (A)
#define ntohs(A)  (A)
#define ntohl(A)  (A)

#elif defined(LITTLE_ENDIAN)

#define htons(A)  ((((A) & 0xff00) >> 8) | \
                    ((A) & 0x00ff) << 8))
#define htonl(A)  ((((A) & 0xff000000) >> 24) | \
                   (((A) & 0x00ff0000) >> 8) | \
                   (((A) & 0x0000ff00) << 8) | \
                   (((A) & 0x000000ff) << 24))
#define ntohs     htons
#define ntohl     htohl

#else

#error "One of BIG_ENDIAN or LITTLE_ENDIAN must be #define'd."

#endif
```

address" (also called the MAC address[3]) come from? Unlike the IP address, this is not something you can request from a local server. In fact, the RARP server referred to in the above quote can only respond to your system by using the hardware address it sent within the IP address request. The hardware address must be built into your system at manufacturing time.

This leads, of course, to two questions. One is: what value should you use for the hardware address? The other is: how do you have that value "built into your system"? I'll answer the second, and easier, question first.

Every system on a physical network like Ethernet or Token Ring includes a peripheral called a *network controller*. This chip is the processor's interface to the physical communications medium. As part of its initialization, the network controller must be fed a unique hardware address. In the case of Ethernet, the hardware address is a 48-bit value. In all but the rarest of circumstances, the hardware address will reside in an on-board non-volatile memory device.

Since the hardware address is unique for each board that rolls off the assembly line, it's typically not placed in the same ROM as the firmware. That way, identical copies of the firmware can be mass produced at a lower average cost per chip, while the unique hardware address can be placed in a very small EEPROM (along with other board-specific parameters, if any) when the board is assembled. The contents of this EEPROM might be determined by the board's serial number and created as part of the manufacturing process.[4]

Now onto the remaining question.

OUI, oui!

Okay, so you know what the hardware address is and what to do with it once you've got it. But where does this globally unique 48-bit identifier come from? Clearly 2^{48} is an incredibly large number of unique addresses, but no set of numbers is itself large enough to guarantee global uniqueness on its own. In the case of Ethernet, the upper three bytes of the 48-bit MAC address are carefully controlled by the IEEE. The scheme for guaranteeing uniqueness of the lower three bytes is individually up to the companies that make Ethernet-capable devices. All of us must work together with the IEEE to guarantee global uniqueness of our MAC addresses.

Every company that wants to produce Ethernet-capable products must first register with the IEEE (*standards.ieee.org/regauth/oui/*). You start by filling out some paperwork and sending them a check for $1,250. In return, your organization becomes the proud owner of a 24-bit Organizationally Unique Identifier (OUI) and the 16,777,216 unique hardware addresses it is capable of identifying.[5]

What I've found usually works best for internal management of the lower 24 bits of the address is to break those bits into two fields. For example, you might break up your approximately 16.8 million unique addresses into 256 blocks of 65,536 addresses. That way, you could allocate the blocks one at a time to particular manufacturing groups, on an as-needed basis. The lowest 16 bits of the MAC address could then match up with the lowest 16 bits of the device's serial number for ease of manufacturing. Following

this scheme, the 255th board manufactured within block two would be given the MAC address "OUI:02:00:FF," where OUI is the three-byte hexadecimal value assigned to your company by the IEEE.

What else?

This month in the Internet Appliance Design section, we've got a very timely set of articles for you. The first, by Greg Wickham, tells us about one company's early experiences with the Java programming language. In this case, the company is Alcatel and the Java code is compiled to native code as part of the development process. In other words, they're using Java without a Java Virtual Machine. Stay tuned folks, I believe ahead-of-time compilation like this has the potential to revolutionize high-end embedded systems development within just a few years.

On the subject of wireless communications, we've got an article about IrDA, by Charles Knutson. This article drills right down into the IrDA protocol stack, showing what you need to implement to support infrared communications with PCs, printers, and other systems. The author focuses on the use of IrDA in resource-constrained embedded systems by showing us what can be left out.

Next month I'll tell you everything you ever wanted to know about the mappings between IP addresses and Ethernet addresses and the corresponding Address Resolution Protocol (ARP). In the meantime, stay connected... **esp**

Michael Barr is the technical editor of Embedded Systems Programming. *He holds BS and MS degrees in electrical engineering from the University of Maryland. Prior to joining the magazine, Michael spent half a decade developing embedded software and device drivers. He is also the author of the book* Programming Embedded Systems in C and C++ *(O'Reilly & Associates). Michael can be reached via e-mail at mbarr@cmp.com.*

References

1. Frankly, I can never remember which is which: big endian or little endian. So I tend to use the terms "human readable" and "not human readable" on a day-to-day basis, especially when I'm troublesh6ooting problems alongside other engineers who may be even less versed in the formal terminology.
2. Of course, a convention must also be agreed upon for the application-level data payload being transmitted over the network. However, in this case it is up to the designer(s) of the application to choose the endianness. They'll usually tend to make the choice that minimizes byte reordering.
3. MAC is short for media access control.
4. It should be clear that there is nothing to preclude your storing the hardware address in any nonvolatile memory device, including the ROM or Flash that contains the firmware. This is a decision, however, that should be made with manufacturing concerns, and costs, in mind.
5. I was shocked to learn recently that Cisco owns 60 such sets, for a grand total of over 1 billion addresses!

Resources

If you're working with TCP/IP, there are two series of books that you should know about. The first is a series of four books by Douglas Comer. In theory, you only need to buy three of these, since there are two versions of Volume 3, but some cross-platform developers may want to get all four. The first volume of the book gives you the big picture of TCP/IP network architecture and the relationships among and between the various protocols. Volume 2 is an implementer's guide, with the most recent edition geared toward ANSI C. The third volume focuses on application-level programming, where you're using the TCP/IP protocols through either the UNIX or Windows sockets API.

Comer, Douglas E. *Internetworking with TCP/IP—Volume 1: Principles, Protocols, and Architecture, 4th ed.* Englewood Cliffs, NJ: Prentice Hall, 2000.

Comer, Douglas E. and David L. Stevens. *Internetworking with TCP/IP—Volume 2, ANSI C Version: Design, Implementation, and Internals, 3rd ed.* Englewood Cliffs, NJ: Prentice Hall, 1998.

Comer, Douglas E. and David L. Stevens. *Internetworking with TCP/IP—Volume 3, Windows Socket Version: Client-Server Programming and Applications, 1st ed.* Englewood Cliffs, NJ: Prentice Hall, 1997.

Comer, Douglas E. and David L. Stevens. *Internetworking with TCP/IP—Volume 3, BSD Socket Version: Client-Server Programming and Applications, 2nd ed.* Englewood Cliffs, NJ: Prentice Hall, 1996.

The second series of books is by W. Richard Stevens (no relation to David L.). The first two volumes of these are my personal favorites, despite the fact that I originally cut my teeth on TCP/IP at a time when only Comer's first edition books were available. This three-volume set follows a similar flow to the Comer books. However, I haven't found as much use for Stevens' Volume 3 as I have for Comer's.

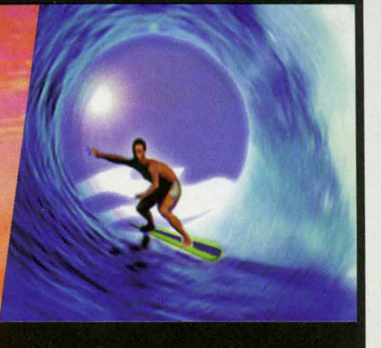
Stevens, W. Richard. *TCP/IP Illustrated—Volume 1: The Protocols*. Reading, MA: Addison-Wesley, 1994.

Wright, Gary R. and W. Richard Stevens. *TCP/IP Illustrated—Volume 2: The Implementation*. Reading, MA: Addison-Wesley, 1995.

Stevens, W. Richard. *TCP/IP Illustrated—Volume 3: TCP for Transactions, HTTP, NNTP, and the Unix Domain Protocols.* Reading, MA: Addison-Wesley, 1996.

In addition to acquiring at least the first two volumes of one of these series of books, embedded programmers who are new to the world of TCP/IP should also familiarize themselves with the online RFCs. The Internet protocols have always been defined and redefined through written Request for Comments (RFCs). To search the database of current RFCs, try *www.faqs.org/rfcs/*.

WE CALM THE SEAS OF TECHNOLOGY



SO YOU
CAN RIDE
THE WAVE
OF SUCCESS

WHY PACIFIC SOFTWORKS ?

• **Protocols:**	TCP/IP, UDP/IP, DHCP, BOOTP
• **Applications:**	FTP, TFTP, TELNET, DNS, SMTP, POP3, RPC
• **Internet:**	PPP, SLIP, CSLIP, CHAP, PAP
• **Web:**	Web Micro Browser, Web Server, Graphics Toolkit/GUI
• **Network Management:**	SNMP v1/v2/v3, MIB Compiler
• **Routing:**	RIP/RIP2, OSPF, BGP4, NAT
• **Security:**	SSL Support
• **RTOS:**	ThreadX™

Since 1982, Pacific Softworks has been delivering the quickest and most stable Internet Protocols available in the industry. We're proud of our reputation as the leading supplier of protocol software.

Royalty Free or Royalty based licensing
End-to-End support from the Protocol Specialists
Ported and Tested Compatibility with any Processor, Kernel or RTOS
Flexible and Scalable solutions to protect your investment
Demonstrated reduction of time to market
Leading the industry with next generation products and solutions
World-wide presence
...and, most importantly, we have Happy Customers!

Visit our website for more in-depth information on Fusion, the "off-the-shelf" FastTrack and other leading embedded solutions. www.pacificsw.com , e-mail: sales@pacificsw.com

Pacific Softworks provides a single supplier solution ensuring that today's successes are the building blocks for tomorrow.

California Corporate Headquarters:
tel: 800.541.9508 fax: 805.499.5512
European Headquarters:
tel: 44.1494.432.735 fax: 44.1494.432.728
Japan/Asia-Pacific Headquarters:
tel: 81.3.5669.7722 fax: 81.3.5669.7723

Pacific Softworks
our wave... your Success.

ALEXANDER WOLFE

Alliances Drive Embedded Linux Toward Prime Time

A plethora of products is due to hit the market soon, even though the Unix-like operating system still has to prove its mettle in the world of Internet appliances.

One of the more interesting Internet appliances I've heard of—a standalone Internet radio which connects via 56k modem—will make its appearance sometime this summer.

While the ability to listen to baseball games—or even country music—far from its broadcast source is useful, perhaps even more salient is the fact that Kerbango Corp.'s appliance is implemented using embedded Linux. "Development was a lot harder when we started a year ago than if we'd started today," said Curt Hewitt, chief technology officer at the Cupertino, CA startup.

Kerbango's stab at applying a downsized version of the traditionally hefty Linux OS to the world of embedded appliances may be an early indicator of what's being replicated at dozens of software developers. That's because purveyors of operating systems and tools are just beginning to roll out a rash of products billed as either "embedded" or "real-time" Linux. According to Kerbango's Hewitt, that wasn't the case a scant 12 months ago, when work on embedded Linux platforms was at a far earlier stage industrywide.

Essentially a clean-room version of Unix, Linux was conceived by then-University of Helsinki computer science student Linus Torvalds. (Today, he's an employee of Silicon Valley startup Transmeta Corp.; see "The Software Side of Crusoe," April 2000, pg. 85.) Like Java, Linux initially made its appearance in the bloated world of Enterprise computing. Lately, however, a plethora of embedded Linux implementations and toolsets are beginning to make their way to market.

"I think Linux is in a footrace," said Michael Tiemann, chief technology officer of RedHat Inc. He pointed to VxWorks from Wind River, Windows CE from Microsoft, and the BeOS from Be Inc. as some of the contenders.

Indeed, most industry experts agree that the next year will be a telling one for Linux in the embedded arena.

Outside operating systems apart, at least five and perhaps as many as 10 vendors within the Linux community itself (in this often secretive corner of the software world, no one is quite sure of the exact number) are going embedded systems code one better and attempting to roll real-time versions of the OS—even though the basic kernel was not designed to support such operation. (Included among that number are Lynx Real-Time Systems Inc., Lineo Corp., MontaVista Software Inc., QNX Software Systems Ltd., and others who either haven't fully disclosed their real-time plans or could not be reached at press time.)

"You have to understand that this is an open source piece of software and people each have their own enhancements," said David Ditzel, president and CEO of Transmeta.

And because Linux is "open source"—a concept pioneered by the Cambridge, MA-based Free Software Foundation—which is famous for its GNU compilation and editing tools, any Linux updates become (theoretically, anyway) available to the entire software community.

In many ways, today's rush toward Linux by a herd of companies is reminiscent of Java two years ago. That is, technical efforts are taking center stage while business models—how to make money—seem to be something of an afterthought.

"It's so early," said James Ready, president and CEO of MontaVista Software Inc. "The wheat will eventually get separated from the chaff both in terms of real technical capability, both in the product itself and in engineering services. In other words, does the company have an understanding of Unix and real-time issues."

Another big differentiator is that Java had a major corporation behind it (albeit one that was largely pushing an Enterprise version of that platform). Linux's biggest corporate support comes from IBM Corp., which has pledged to support the OS in everything from full-blown mainframes all the way down to Internet appliances. To help the company deliver on that promise, IBM recently formed an internal Linux group.

However, most of the names heard on the Linux battlefield are either new companies or the traditional purveyors of RTOSes. And even there, many of the old-time RTOS players are taking a careful wait-and-see attitude before they divert resources from their primary, proprietary OSes and jump full-tilt into the Linux fray.

Indeed, most industry experts agree that the next year will be a telling one for Linux in the embedded arena.

"We are seeing a serious groundswell in terms of using Linux in embedded systems," said Doug Locke, president of TimeSys. "I'm talking about a wide range of applications, stretching from very small things one might find in an automobile to much larger military-control apps."

There's just one major stumbling block on the road to a Linux that's useful for developers of all stripes. Namely, Linux can clearly be better downsized to fit the bill for tight-footprint embedded applications. However, whether its kernel can be extended to deliver real-time deterministic performance is another question entirely.

At issue: the native Linux kernel is designed for coarse-grained synchronization and system tasks can't be preempted.

Kernel characterization

Still, several companies are laying down bets that they can deliver real-time versions of Linux. Such an OS would obviously have a jump on an embedded-only offering, which would at best support "soft" real time or, if not that, then "human" time operation such as might be required in an automotive entertainment system.

Two of the more notable participants in this rapid rush to real time are newcomers Transmeta Corp. and MontaVista Software Inc.

In that regard, a question had surrounded Transmeta Corp., which earlier this year released two embedded processors powered by a downsized version of Linux. At the time, it wasn't clear whether the company would open up its "mobile Linux."

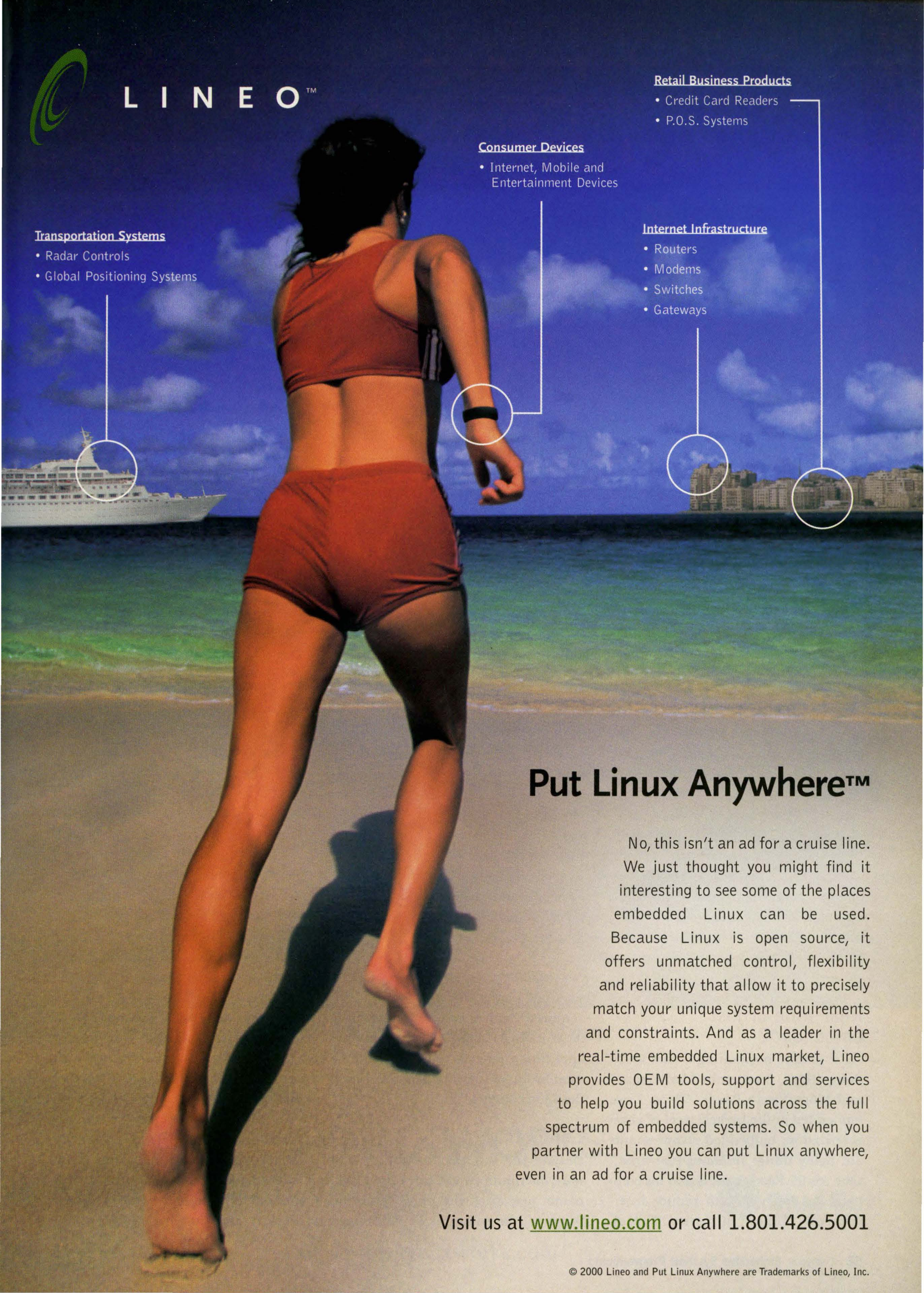
Ditzel of Transmeta intends to put those questions to rest. "Our mobile Linux will absolutely be open source," he said. "What we're doing is, number one, adding power management and number two, adding a compressed file system called CRAMFS (for 'cram' file system)."

This enables programmers to shoehorn files into about half the space they would normally require. "In an [Internet] appliance, where you might want to store your program in a flash ROM, rather than using a hard disk, you'll need only half as much flash ROM—and that's a big deal," he said.

Contenders abound in what may be an early sign of polarization in the Linux marketplace. But one of the big questions is just exactly what the memory footprint must be, at minimum, to meet bare minumim embedded requirements. Tiemann of RedHat floats a figure of 4K as his low-end target. However, such a setup offers very limited funtionality—basically a kernel with no file system, no networking support, no user interface (which must be tacked on separately), and only bare-bones I/O.

Indeed, others disagree that a 4K profile is capable of providing what will be required for a total solution. (Even Tiemann's breakout of the user interface as a separate number would seem to validate such a conclusion.)

More than anything else, Linux's legendary size—it has hundreds of services and calls—has made it a bear to wrestle to real-time ground.

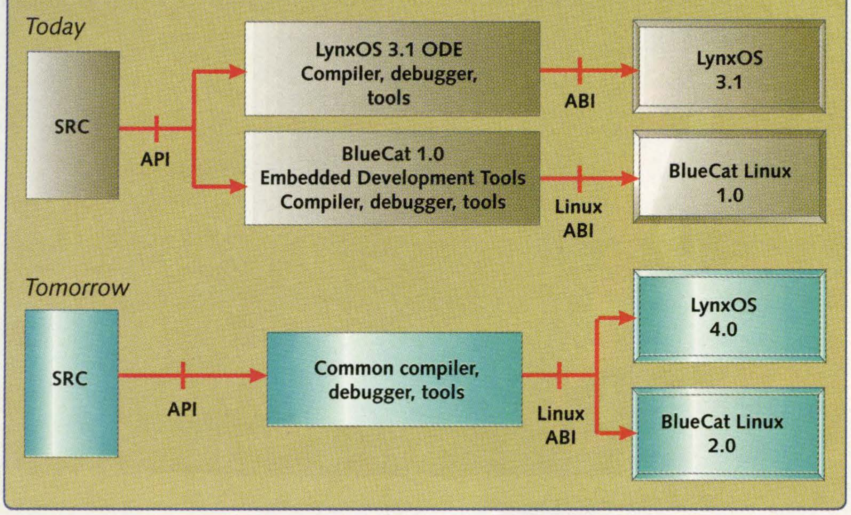


Put Linux Anywhere™

No, this isn't an ad for a cruise line. We just thought you might find it interesting to see some of the places embedded Linux can be used. Because Linux is open source, it offers unmatched control, flexibility and reliability that allow it to precisely match your unique system requirements and constraints. And as a leader in the real-time embedded Linux market, Lineo provides OEM tools, support and services to help you build solutions across the full spectrum of embedded systems. So when you partner with Lineo you can put Linux anywhere, even in an ad for a cruise line.

Visit us at www.lineo.com or call 1.801.426.5001

FIGURE 1 Lynx Real-Time Systems will use its "Blue Cat 2.0" offering to deliver hard real-time Linux support


Today
SRC
API
LynxOS 3.1 ODE Compiler, debugger, tools
ABI
LynxOS 3.1
BlueCat 1.0 Embedded Development Tools Compiler, debugger, tools
Linux ABI
BlueCat Linux 1.0
Tomorrow
SRC
API
Common compiler, debugger, tools
Linux ABI
LynxOS 4.0
BlueCat Linux 2.0

"To do anything really useful including a kernel and reasonable functionality, you have to go to at least half a megabyte," said Ready of MontaVista. "Much smaller than that is not going to be interesting." However, he notes that more complex systems, such as those that run multiple applications, will have to have even bigger footprints. For example, adding a sophisticated file system could boost things to the 4MB range.

Kerbango, which has used MontaVista's HardHat Linux in its Internet radio, says the kernel, TCP/IP stack, and support for virtual memory mapping require about 600K. The user interface and software codec that enable the radio to play MP3 audio files aren't included in that count.

Real-time or just embedded?

"Our unique positioning is that we're doing 100% pure embedded Linux," said MontaVista's Ready. "Here, I use embedded as the highest level of characterization; embedded requirements are many things *including* real time." One result, Ready said, is that his company has been able to reduce interrupts-off times in the kernel. MontaVista is now shipping that feature and also has folded it back into the open-source version of Linux.

Indeed, the most serious functionality requirement that separates the "hard" kernels from their pokier embedded cousins is true real-time. That's been a subject of much debate in the Linux community and it's not a subject likely to be resolved anytime soon.

"To make it real-time, what you have to do, fundamentally, is deal with memory, CPU, and I/O resources in such a way that when you 'speak' with them for parts of the system that have to share those resources, they can actually meet their time contraints," said Locke of TimeSys. "Linux as it exists doesn't really do that."

Another route to hard real time is coming from Lineo Corp., which acquired Zentropix Computing earlier this year. The latter's real-time application interface (RTAI) will be used by Lineo to bring determinism to the Linux kernel.

Competing flavors

Despite the fact that numerous software distributions will carry the Linux name, peel back the onion skin and there will be some major differences between the implementations making their way to market.

Unlike Java, Linux is not driven by a group of developers intent on forging a firm standard—despite the fact that it sometimes seems that way. Strictly speaking, Linux is open-source code. Perhaps that's why there seems to be reasonably cooperative competiton in the Linux community. That stands in marked contrast to Java, where Sun Microsystems holds all the cards. (Of course, competition could get far more intense once products start to hit the market in force.)

"I think the Linux community is much more diverse than anybody realizes," said Tiemann of RedHat. "And there are people connected with Linux who are real embedded gearheads."

Regardless, sometimes standards are in the eye of the beholder.

"Using the word 'standard' feels funny, because it connotes something that is not the way Linux works," said Ditzel of Transmeta.

"Some people would like to have their particular ideas as a standard because then they have other products surrounding that, which will fit nicely," said Paul Zorfass, analyst at First Data Inc.

For one, RedHat Inc. is pushing ahead in its bid to nail down its homegrown Linux application programming interfaces (APIs)—called EL/IX—which it hopes will come to become dominant.

Tiemann of RedHat doesn't see anything nefarious in that. "The underlying principle that [Linux creator] Linus Torvalds preaches is that the best code wins," Tiemann said.

EL/IX, he explained, is an attempt to pare down Linux so that it's manageable for embedded apps. "We took the APIs that Linux programmers use on a daily basis—for example, printf is from the C library, sleep is from the kernel—and we looked at the Posix 1003 standard and we looked at how that standard defines bundles of functionality and how it defines applica-

ORCHESTRATING EMBEDDED LINUX SOLUTIONS

It's been true in music since the first duet. And today, it's just as true in the world of Linux. A technology may play beautifully, but for a solution to be complete, this technology must also play in harmony. With compelling complementary technologies. With worldwide, world-class consulting, support, and training services. Without one false note.

That's what we at LynuxWorks bring to the performance. As Lynx Real-Time Systems, our company developed LynxOS® RTOS, a true virtuoso in developing embedded real-time applications and the closest RTOS to Linux today. Now, with BlueCat™ Linux, we've expanded our repertoire to embedded Linux. Embedded systems designers can use either of these superb soloists, or tap the natural synergies of both, to speed new product development and more precisely match product and market requirements.

To mark this change, our company has taken a new name. One that integrates our past with our future. And, like a seasoned maestro, we've brought all our resources – experience, technology, and support – with us.

Don't miss a beat. Find out more about LynuxWorks. The company that orchestrates embedded Linux solutions. The company that can help you hit new high notes in embedded systems design.

Lynx Real-Time Systems is now

LYNUXWORKS™

LynxOS FROM LYNUXWORKS

BlueCat™ FROM LYNUXWORKS

www.lynuxworks.com

As with the Java world, the big question remains of how the Linux players plan to make money selling a product that typically ships for under $100.

tion and operating system profiles," Tiemann said.

"We created a definitive list of ways that you can slice and dice all the entry points in these various APIs," Tiemann said. "So that you could, from a top down perspective, define a set of functionality that was consistent between two different people working from two different source [distributions]."

RedHat's EL/IX draft is available on their Web site. Their EL/IX distribution will be available by mid-fall, Tiemann said.

Separately, Ready of MontaVista noted that he took a different approach, by helping to found the Embedded Linux Consortium. What's the purpose of that group? Linux promotion, pure and simple, he says.

For real-time developers keeping their eyes focused on the open source distribution, the Web site of choice remains *www.rtlinux.org*. The central repository for Linux remains the *kernel.org* site.

Making a dime

The final question for developers—though perhaps the first one for marketers—is just how Linux's purveyors are going to make a profit. As with the Java world, the big question remains of how the Linux players plan to make money selling a product that typically ships for under $100.

"It's not on the operating system," said Darrin Skewchuk, an executive at QNX. "It's on the support and services."

Indeed, some stalwarts of the RTOS world are waiting to see whether Linux will make as big a splash as some expect. "Right now, Wind River is genuinely trying to determine the appropriateness of Linux for the embedded space,"said Curt Schacker, a vice president at the company.

"Embedded is a lot of things," said Schacker. "It's very broad, there are many different applications, and I think that Linux has gotten attention in some of those spaces—most particularly in the Internet appliance space where people are interested in using the Intel x86 architecture."

Ready of MontaVista believes Linux could prove complementary to Java. "We're finding that Linux is becoming a very strong platform to host Java," he said. "That is, you don't want to run Java on a bare machine, so you have to run it on something. And we're finding extreme interest in Linux being the engine for Java. We're working with three or four Java implementations right now—that's a huge trend."

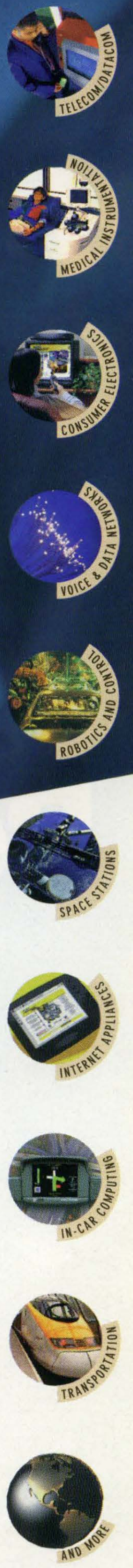
For now, however, Schacker of Wind River plans to keep taking reality checks on the software flavor-of-the-month. "I think a very important question is, will Linux find acceptance in the broader embedded space. Is it going to move into networking, industrial control, and military/aerospace," he said. "I don't think there are answers to those questions yet. So what we're trying to do is study the trends and understand where Linux is going to go." **esp**

Alexander Wolfe is managing editor for microprocessors and embedded at Electronic Engineering Times. *He holds a B.E. in electrical engineering from Cooper Union. He wrote assembly language code for embedded systems in the early 80s. He later co-authored* From Chips to Systems: An Introduction to Microcomputers—2nd Edition *(Sybex, 1987). He can be reached at awolfe@cmp.com.*

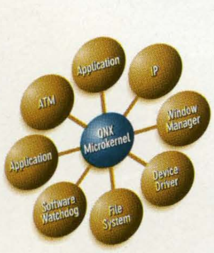
Change the way you think.

Think what you could change.

BUILD A MORE RELIABLE WORLD™



START WITH A MORE RELIABLE OS



It goes against conventional thinking. But to build more products in less time — without sacrificing reliability — you need more than good tools. You need a better OS.

An OS with an architecture so advanced it lets you pinpoint memory faults in minutes. Prototype without system rebuilds. Add features without retesting your entire software suite. Upgrade systems without rebooting. And respond faster to what customers want.

Now imagine. What if this same OS was backed by two decades' experience in the embedded market. An outstanding reputation for technical support. And an OS company as reliable as its technology. Think how that could change your world, and your customers' world.

For the better.

www.qnx.com

Call about our eval system:
800 676-0566 ext. 2325

QNX® RTOS

Build a more reliable world™

Small Footprint...
Huge Impact!

Extending data beyond the corporate enterprise. PointBase, the first pure Java data management solution designed to support applications for the mobile and wireless device market.

PointBase

Managing data ***anywhere, anytime***

Meet the demands of the small application environment...test-drive the PointBase data management solution with synchronization... discover the keys to high-power, small-footprint databases...pick-up your FREE 30-day evaluation copy now!

www.pointbase.com/esp

Questions? Give us a call 877.238.8798 toll-free

GREG WICKHAM

Compiled Java Experiences

Embedded software developers who want to use the Java language have several options, the best known of which is the Java virtual machine. However, over the past few years several compiler vendors have made good progress in developing cross compilers for Java. This article looks at the advantages of compiled Java and offers advice to embedded programmers new to the Java language.

It's been reported that Java is 10 to 20 times slower than C, on average. Accurate or not, those figures are attempting to describe "interpreted Java." In other words, if you employ a Java virtual machine in your system to interpret Java bytecode on the fly (instruction by instruction), you will pay a serious run-time performance penalty. Such a large performance penalty, combined with the ROM and RAM requirements of the embeddable JVMs available today, is likely an insurmountable obstacle to using Java in the majority of embedded systems.

However, I have recently tried using Java in another way. Using a traditional embedded software development model and a Java cross compiler (sometimes called an ahead-of-time, or AOT, compiler), I have been developing part of my embedded software in Java. Following this scenario, my embedded system has no JVM at all, and the execution speed of programs written in Java compares favorably with those written in C++.

Development tools

If you're contemplating bringing Java into a development shop dominated by Solaris, you should know about one of Java's supreme ironies: most of its developers are Windows-hosted. This means that virtually all of the major IDE vendors produce a Windows version of their latest release first, with the Solaris port trailing behind by six months or more. The same principle applies to vendors of other Java products, like performance analysis, testing, and CASE tools.

In general, Java development tools targeting the embedded market are not as mature as those targeting desktop developers. Most embedded tool vendors are building on top of C-based legacy technology, and they often have less experience with Java than comparable vendors of development tools for the desktop. I've found it useful to compare embedded Java tools side-by-side with their desktop counterparts. By knowing what's available in the top-of-the-line tools, you can more wisely choose the features that really matter to you and make sure your embedded tools have them.

Because Java is designed for portability, it's often possible to develop, test, and debug significant portions of an embedded application in a feature-rich desktop environment. I do this by putting "interface objects" at the boundaries of my Java code. These interface objects are responsible for services like timing and low-level I/O. On startup, the system installs only the set of interfaces that is appropri-

In a conventional (JVM-based) Java runtime environment, classes in the base libraries are written in Java, just like all the other classes.

ate to the environment in which it finds itself, rather like plug-and-play device drivers. Our current system uses four such configurations—one each for development, simulation, emulation target hardware, and "real" target hardware. The configuration can also be fixed at compile time by using compiler conditionals or simple changes to the source.

Java libraries

In a conventional (JVM-based) Java runtime environment, classes in the base libraries are written in Java, just like all the other classes. The developer has access to the base class source code, and can modify or enhance it as needed.

Statically linked Java doesn't necessarily work that way. The base class libraries that come with an AOT compiler might be written in a lower-level, higher-performance language. They might not be the least bit object oriented on the inside. But the base classes only need to look object oriented from the outside. What goes on behind those method entry points is really none of the Java programmer's business anyway, so long as the classes work as advertised.

Some Java libraries like `java.io` and `java.net` make extensive use of platform services through native methods (calls to functions written in C or assembly). It's possible to port a base class to a new platform by simply implementing each of its native methods on the target platform, and using an AOT compiler on the Java code. But this may not be the best approach, for at least two reasons:

- Base libraries made from compiled Java code probably won't be as fast as they could be. These libraries may have been written with portability as the primary requirement
- Vendors that use Sun's Java code to make their base libraries will have to pay license fees to Sun. It probably makes more business sense for them to develop their own ("clean-room") libraries

Libraries built from Java code will probably be the most flexible. What's going to happen to those optimized clean-room libraries when Sun comes out with the next version of the JDK and it has changes in the base classes? Is the compiler vendor going to update that hand-hewn code to reflect the changes? How soon? Will a project get stuck using an older version of Java—including older language constructs—just because the new Java libraries haven't been ported yet? For projects that only want to use the core features of Java, it probably isn't as important to keep up with the latest JDK version. So this may be less of an issue for developers using AOT compilers.

The structure of base libraries is also a factor when eliminating dead code in order to minimize footprint for application deployment. Base classes built from compiled Java can be recompiled to exclude classes and methods that are not used by a given application. Clean-room libraries should also provide some means of excluding code that an application doesn't need.

Build process

In my organization we use RCS for revision control of our C code. The source files to build any given C-based system all reside in a single directory, and our staff have developed a rich set of scripts and utilities for managing the source within this environment.

The package-structured nature of Java code is fundamentally incompatible with this system. To permit reuse of packages as components, Java source code needs to be organized into a directory tree whose structure parallels the package hierarchy of the code. The make process must be able to select components from the source tree that are to be included in a given build. For my group this means that the staff have had to develop new scripts and processes in order to incorporate Java into our traditional build processes.

The AOT compiler we are currently using comes with a "driver" program that can essentially manage the entire build process, at least for wholly Java applications. One or more of the classes in the source code tree need to have `static main()` methods, just as if we were building a standalone Java application targeting a JRE.

The compiler's "driver" program compiles the root source file first. Any references to symbols outside a class's own package must be resolved at compile time through `import` statements in the code. The driver works its way through the transitive closure of method calls from `main()` in the root class, compiling each module and statically linking them all together into a single object module.

This one-step compile-and-link procedure is a simple way to get a wholly Java application up and running, but it doesn't provide the flexibility needed to build real-world embedded systems. Specifying different compilation options for different files is likely to require the use of a traditional makefile, especially for systems that also include code written in other languages.

Note that static linking precludes some of the really fancy things that can otherwise be done with Java's reflective capabilities. For example, it is not possible to construct the name of a class at runtime and then "load" the class. Depending on the way a given compiler handles class metadata, certain portions of the `java.lang.reflect` API may not be available.

RTOS integration

The next step is to link the object module generated by the Java compil-

Vision available. Hardware not included.

ESPIAL™

BUILD A SMART FUTURE™

espial.com

Systems that don't use a garbage collector can eliminate the collector itself, thereby reducing the memory footprint.

er into an image that can be run on a given target platform. This is where, for example, the creation of a threaded object:

```
class   AuditDaemon   implements
Runnable
{
  public void run()
  {
    //System audits go here
  }
}

public AuditDaemon auditor = new
AuditDaemon();
```

is mapped to the creation and scheduling of an OS task. To improve portability, some Java compilers target a standard threading API like POSIX pthread.

The most common way of accessing low-level resources from Java is the use of native method. These look like any other method to their clients in the Java world, but they are implemented in a language other than Java—usually C or assembler. Inside a native method, your code is not subject to any of Java's security restrictions or runtime error checking, so you have unrestricted access to platform resources.

A Java toolset for embedded development might also come with libraries that give the developer access to machine resources in a convenient way. For example, at least one comes with a set of classes that provide simple, object-oriented access to regions of target memory.

Garbage collection

Most desktop Java systems use a blocking garbage collector (GC), which means that when the garbage collector runs, all Java threads stop until it's finished compacting the heap. For applications that can tolerate this behavior, a blocking collector probably offers the best average performance.[1] However, for many embedded applications that have even soft real-time requirements, blocking garbage collection is just not acceptable.

One alternative is to not use a garbage collector at all. Some specialized Java libraries support an operation similar to `free()`, which allows the programmer to deallocate an object's memory in the conventional C/C++ programming style. Even if your tools don't support this, it may be possible to preallocate all of the memory that will be used by an application, making the garbage collector unnecessary. This could mean designing the system to use only permanent objects, or reusing unneeded objects by returning them to pools for reallocation. Systems that don't use a garbage collector can eliminate the collector itself, thereby reducing the memory footprint.

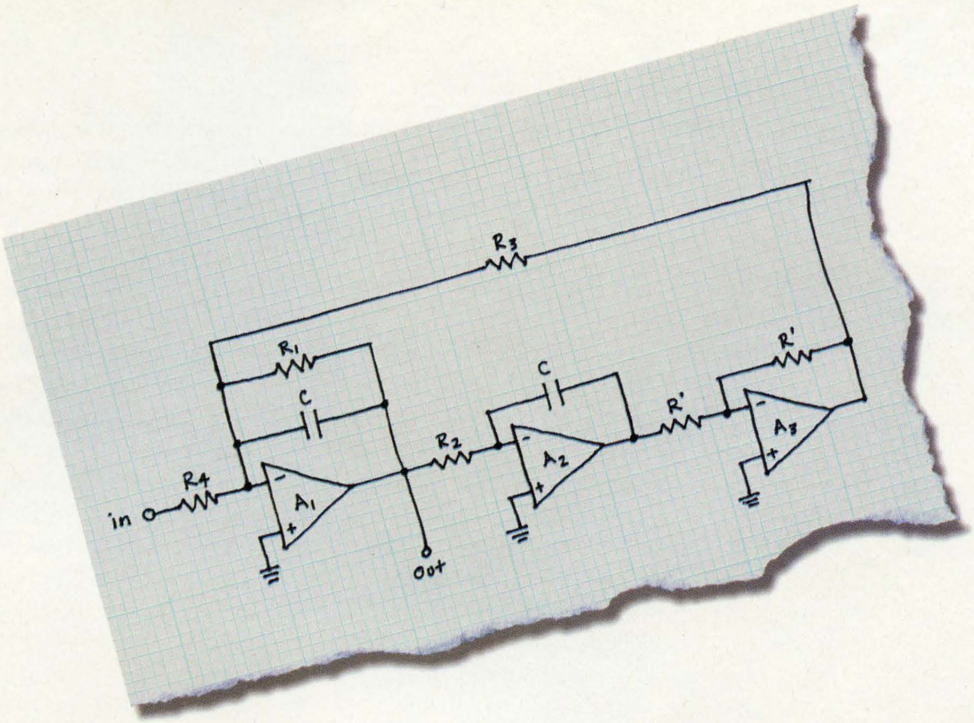
Is it possible to have both the convenience and safety of a garbage collector *and* some measure of runtime determinism? Yes, certain things can be done, but remember that determinism will cost you something.

Maybe the garbage-collected portion of an application's memory is small enough that the GC can compact the entire heap in less than the desired response time. A blocking GC might be just fine for an application like this. Some garbage collectors allow certain objects to be designated as permanent, meaning that they don't have to be scanned by the collector when it runs. If an application's memory can be partitioned such that only a small portion of it needs to be scanned, the worst-case execution time of the GC can be reduced. It's also possible to invoke Java's GC explicitly, using `System.gc()`. If an application knows that it definitely won't need to respond to an event for a given period of time, perhaps it can seize that opportunity to compact the heap.

Some Java tool vendors offer an incremental garbage collector as an option. With this feature it's possible to bound response latencies by telling the collector to never run uninterrupted for longer than the application's desired response time. Many incremental collectors are also preemptible, meaning that they can be interrupted at will.

That may be well and good, but what happens if a system gets so busy processing interrupts or other events that the GC never gets to run at all? Eventually the application will be unable to allocate more memory, and it will have to let the collector run at least until it can fulfill the current allocation request. To keep this from happening one can increase the priority of the thread or task running the GC, but this is done at the expense of application code. Also, because an incremental collector has to maintain extra state information and do extra work, it will have lower average throughput than a blocking collector. Depending on its implementation, an incremental collector might add a small amount of memory overhead to every object in the garbage-collected heap.

NewMonics' PERC system adds another tool for making Java code deterministic. PERC is based on a runtime executive whose job is to schedule Java tasks. Each task is required to supply information about the resources that it will need in the worst case, including memory and execution time. Before scheduling a new task, the PERC executive examines its current workload and decides whether the proposed task can be accommodated while still meeting all previously promised task completion times. If the new task cannot be accommodated, the executive refuses to accept it for scheduling. Of course the executive adds overhead at runtime, but it does



Analog spoken here.

In this digital world, analog engineers often tell us they feel misunderstood. When the talk around the table is in ones and zeros, they're thinking in terms of currents and voltages. They say it's almost as though if you're doing analog, you're living in a different place. And speaking a different language.

If this is *your* world, we've got some good news: now you have a place you can feel right at home. We call it Planet Analog. And it's built just for you.

Planet Analog is a place on the Web devoted entirely to analog technology. A place where you can get the latest news, in-depth technical coverage and product data. Plus articles contributed from some of the finest technologists in the world. And in coming months, we'll add seminars, chats, polls, conference registration and a lot more. All under the leadership of Steve Ohr, regarded by many as the leading analog advocate in the world today.

So if you're an analog engineer, come to www.planetanalog.com for a place to call your own. You'll find we speak your language.

SPONSORED BY:

Texas Instruments

Analog Devices

National Semiconductor

EDTN NETWORK PARTNERS

EBN, EET, SBN, Embedded.com, ISD Magazine, Power Designers, Home Toys, Electronic Design, Microwaves & RF, EE Product News, Wireless Portable, Wireless System Design, PCN Alert, ChipCenter, Communications System Design, Design & Reuse, Circuits Assembly, HDI, PC Fab, Printed Circuit Design

www.edtn.com

EDTN network

It's all right here.

Getting the most out of object-oriented development means a highly interactive, fast-paced cycle of design-code-compile-link-test.

permit the development of reusable software components with specified real-time characteristics—quite an intriguing notion.

Although garbage collection takes most of the heat for Java's lack of run-time determinism, it isn't the only factor limiting Java's suitability for real-time programming. The Java language spec defines the `synchronized` modifier in such a way that it is not generally possible to predict how long a blocked thread will wait to obtain a lock. An experts group has been chartered under the Java Community Process to define a specification for real-time Java.[2]

Debugging

Getting the most out of object-oriented development means a highly interactive, fast-paced cycle of design-code-compile-link-test. Such a process demands a remote source-level debugger that's well integrated with the Java compiler. Many of the available Java debuggers for embedded development started out as C tools and have only recently been extended to handle Java as well. That's fine, but it's also one of those opportunities to carefully compare a remote target debugger with the one in a state-of-the-art desktop Java IDE. Make sure your host-based target debugger has the features that are important to you.

Here's an example. While evaluating one debugger we found that it would always display the structure of objects on the execution stack according to their type as declared in a given stack frame. That may be exactly what you need for looking at C `structs`, but it's not sufficient for inspecting Java objects whose actual type may be something completely different than their role in the current frame. This was one case where side-by-side tool comparison was useful. Without being accustomed to the debugger in my luxurious desktop IDE, I may not have noticed this missing feature until it was too late.

AOT compilers

Several vendors have jumped into the Java cross-compiler market. These include true AOT compilers, like the FastJ and GCJ products from Wind River Systems and Cygnus, respectively, as well as some non-traditional approaches.

FastJ, from Wind River Systems (formerly Diab Data)
www.ddi.com/products/fastj/
Diab's competitive advantage in the compiler space has historically been its optimizing back-end, which is particularly well-suited to Motorola's 32-bit processors and cores. FastJ simply adds another language front-end to their core C/C++ compiler technology. The FastJ Compiler Suite includes the C/C++ compiler, assembler, linker, and other tools you'll need, and is available for the same set of processor architectures.

GCJ, from RedHat (formerly Cygnus)
sourceware.cygnus.com/java/
The GNU Compiler for Java (GCJ) is a Java extension to the popular GNU compiler (GCC). Like GCC, it is open source software. Since GCJ is simply a new language front-end, Java source or bytecodes can be translated into the native opcodes of dozens of popular 32-bit processors. The same back-end optimizations that are performed on the Java code are performed on all other supported languages. The garbage collector and other run-time support are provided in the separate `libgcj` package, which has been ported to some, but not all, of GCC's supported processors.

ROMizer, from NewMonics
www.newmonics.com
NewMonics focuses on the use of Java in the real-time embedded market. Their competitive edge is in making Java execute deterministically. The AOT compiler included within their ROMizer tool is not responsible for the determinism of their solution. Rather, it provides a way to get more speed out of Java code. Only x86 targets are supported by this AOT compiler.

TurboJ, from Wind River Systems
www.windriver.com/products/html/turboj.html
TurboJ takes a different approach to Java compilation. Rather than compiling and statically linking an entire Java application, TurboJ compiles methods, but still uses a JVM to invoke them. This allows the compiled methods to run at native speed, while still permitting new code (bytecode or native) to be dynamically loaded at runtime. TurboJ converts the Java bytecodes in a .class file (not source code) to an object file that can be linked to the JVM.

TurboChai, from Hewlett-Packard
www.hp.com/emso/products/turbochai/
TurboChai is the AOT compiler for HP's ChaiVM system for embedded Java development. TurboChai turns Java source code or bytecodes into ANSI C code, one class at a time. Like TurboJ, it works only alongside a JVM. In this case, that must be HP's ChaiVM.

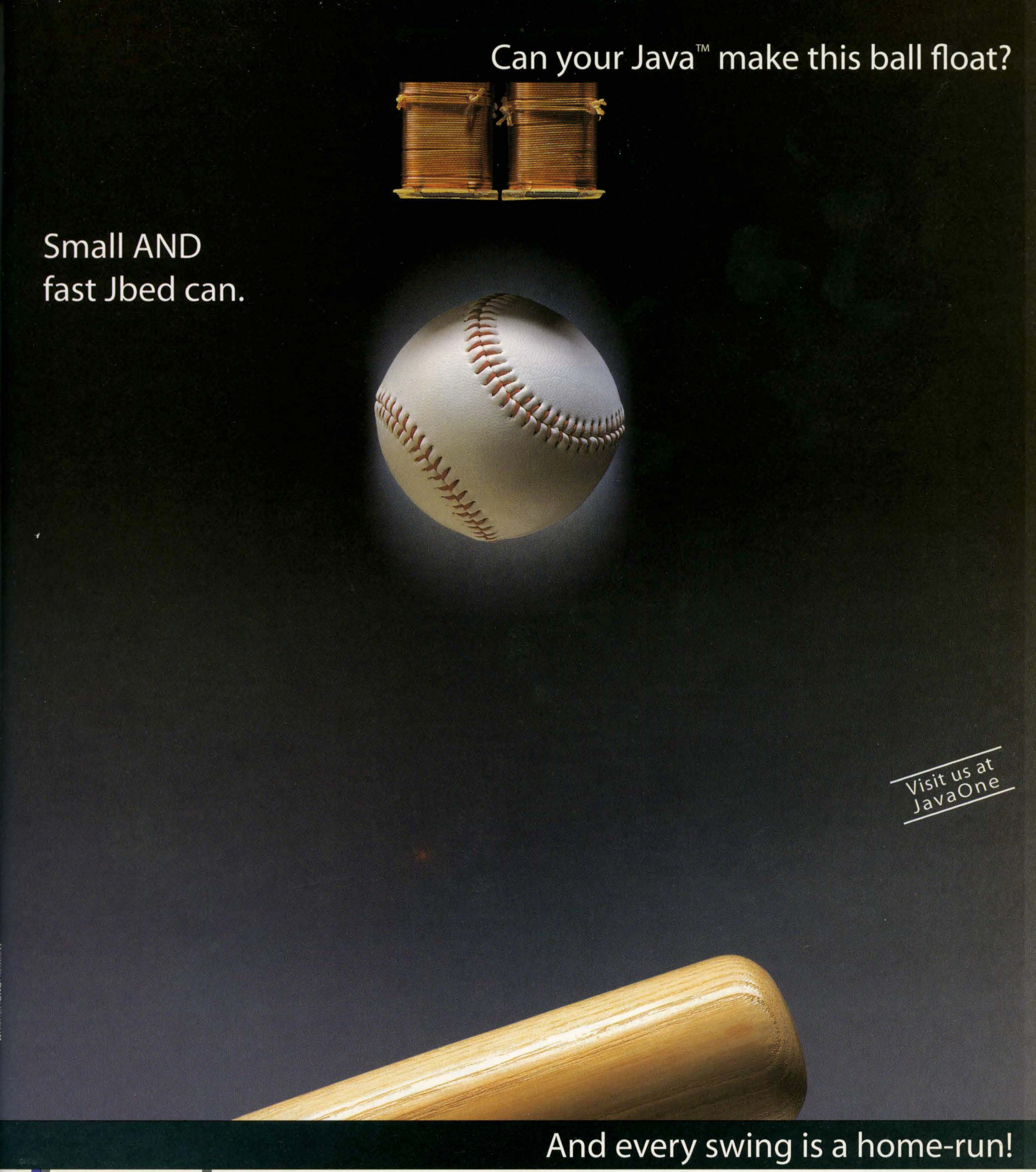

Can your Java™ make this ball float?
Small AND
fast Jbed can.
Visit us at
JavaOne
And every swing is a home-run!

FIGURE 1 Organization of some embedded systems

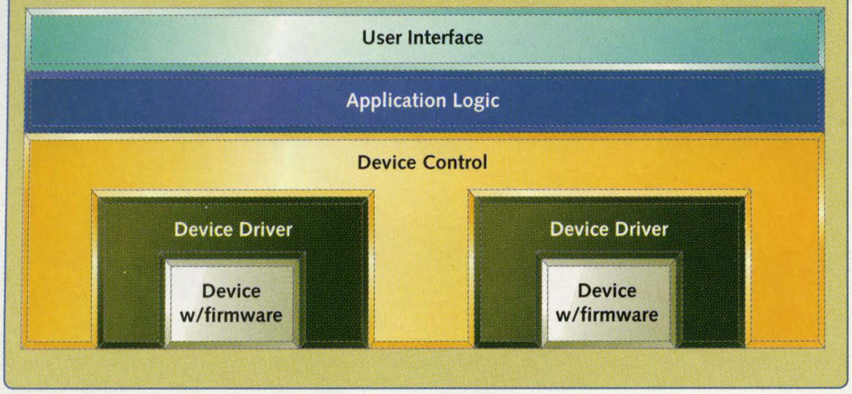


TABLE 1 Possible requirements for the various subsystems

Subsystem	Real-time?	Grasped by many?	Changes rapidly?	Reused elsewhere?
Firmware	Yes	No	No	Probably not
Device drivers	Yes	Maybe, if reused	No	Maybe
Device control	Probably	Maybe	Probably not	Maybe
Application logic	Maybe	Yes	Yes	Probably
User interface	Not really	Yes	Yes	Maybe

Most embedded systems that use Java will probably include some C or C++ code as well. Such mixed-language systems require a debugger that works smoothly with all the languages they employ—usually C, assembler, and Java. In this area, their C legacy may actually give embedded tool vendors a leg up on the desktop world.

High-performance Java

I always try to follow Kent Beck's advice about the optimization process:

- Make it work
- Make it right
- Make it fast

Once you have a properly factored system that meets the bulk of its requirements, you are ready to accurately analyze performance and make improvements that won't wreck your design.

Before torturing your beautiful OO designs in an effort to wring every last instruction cycle out of your Java code, make sure you understand why you're using Java in the first place.

Object-oriented techniques offer programmers some powerful tools for managing complexity in problem domains that are inherently complex. Elegant and well-abstracted designs are particularly important for systems that need to be quickly grasped by many people, that are constantly being revised and extended, or that need to be reused in other contexts—other devices in the same product family, for example. But this isn't really the case for a lot of embedded code, especially code that's "close to the metal." So think twice before deciding to give up speed and space to gain flexibility that you don't really need.

The embedded systems I've worked on tend to be organized as shown in Figure 1. Requirements for the various subsystems look something like Table 1.

The upshot of this model is that it generally becomes more tempting to use OO techniques as you get higher up in the architectural layers. I think the case for OO is strongest in modeling application logic, for the following reasons:

- The user requirements for application logic tend to be the most complex. Anything that improves communication between marketers and developers—like OO modeling—can pay off big
- Application logic is frequently at the heart of the value added by an embedded device vendor. As such, it is likely to be valuable intellectual property that should be made as reusable as possible
- The very act of building a cross-product enterprise data model—particularly one that is accessible to marketers—can suggest new directions for product evolution

Memory management. Just because you have a GC doesn't mean you should completely forget about memory management. Thanks to that garbage collector, you no longer have to track references to a data record across multiple threads. You never again need to worry about crashing the whole system just by freeing an object too early. For all the things it does for you, doesn't the GC deserve just a little bit of consideration in return?

Try to be aware of the kind of garbage being generated by your code, including Java code that you use but that you didn't write. Some very well designed OO code creates and immediately discards a lot of small, transient objects.

Know the characteristics of your GC and make sure it matches the needs of the application:

- Blocking collectors run when available memory dips below a configured threshold. When a blocking collector is running, no other Java thread can use the heap until it is finished
- Incremental collectors spread their activity over time so they never get very far behind
- A preemptible collector doesn't need to run to completion each time it is started
- Defragmenting collectors rearrange

Java's innocent-looking String concatenation operator (the plus sign) allows programmers to write code that looks reasonably efficient but is not.

the heap to maximize the size of free blocks

- Generational collectors partition their heap space into differently managed areas according to object lifecycle information gathered at runtime. Many generational GCs allow the user to configure initial sizes and management policies for their partitions

Exceptions. Java's simple but powerful exception-handling capabilities allow the developer to more precisely specify the external behavior of software building blocks. In my first experience with compiling and embedding Java I got carried away with this idea, throwing exceptions to indicate conditions such as "I found the database location but it's empty. Shall I put a new record there?" The performance looked great running on my 333MHz PC system during development, but on a 25MHz target hardware it was a dog. The profiler showed that about 40% of the time, my too-slow code was performing exception handling. In this case I was able to speed the code way up by using special "undefined" placeholder objects instead of throwing exceptions. Nowadays, I limit my use of exceptions to occurrences that truly are uncommon.

Strings. Java's innocent-looking `String` concatenation operator (the plus sign) allows programmers to write code that looks reasonably efficient but is not. This operator allocates a new `String` whose combined length is the sum of the lengths of the two operands, and copies the contents of the two operands into the result. The operands then become garbage unless other objects still refer to them. All this is not so bad until we write an expression like:

```
string1 + string2 + string3 +
string4 +
```

An optimizing compiler can do a fine job with this if all the operands are available at compile time. If not, however, this expression is going to be evaluated at runtime. It will generate *n*–1 throwaway `Strings`—where *n* is the number of concatenations—and the GC will be working overtime to keep up.

A more subtle form of the same problem can occur when concatenating `Strings` as part of a recursive algorithm. The program may only do one concatenation per recursion, but if the recursion gets deep the garbage can really pile up.

The solution in either case is to use the `StringBuffer` class for multiple concatenations. Create a single new `StringBuffer(initialSize)` whose initial size is large enough to contain the final result, and `append(eachOperand)` to the buffer. When finished appending, the buffer's `toString()` method can efficiently copy its contents into a new `String`.

Device control. High-performance embedded Java code doesn't just have to run fast—it also needs to provide interfaces to code in other languages. If the other code is responsible for device control or synchronous protocols, it probably has more strict timing requirements than the Java code, and its performance had better not be limited by the speed of Java.

This problem came to light in one system where we built a configuration management system in Java, with a device control layer below using the managed database to determine its reactions to real-time stimuli. The shared database is a containment tree of addressable managed objects that can be navigated and queried for the values of their parameters. In our first implementation, the objects stored their parameters, which are typically simple data types, in Java data structures like `Vectors` and `Hashtables`. Client code written in C would gain access to the parameter values by invoking methods on the managed objects such as `parameterNamed(aString)` or `parameterNumber(anInt)`. While this is a fine example of encapsulation and data hiding, it forces timing-sensitive C code to wait for Java methods that look up their result using elegant high-level data structures. Not such a great idea.

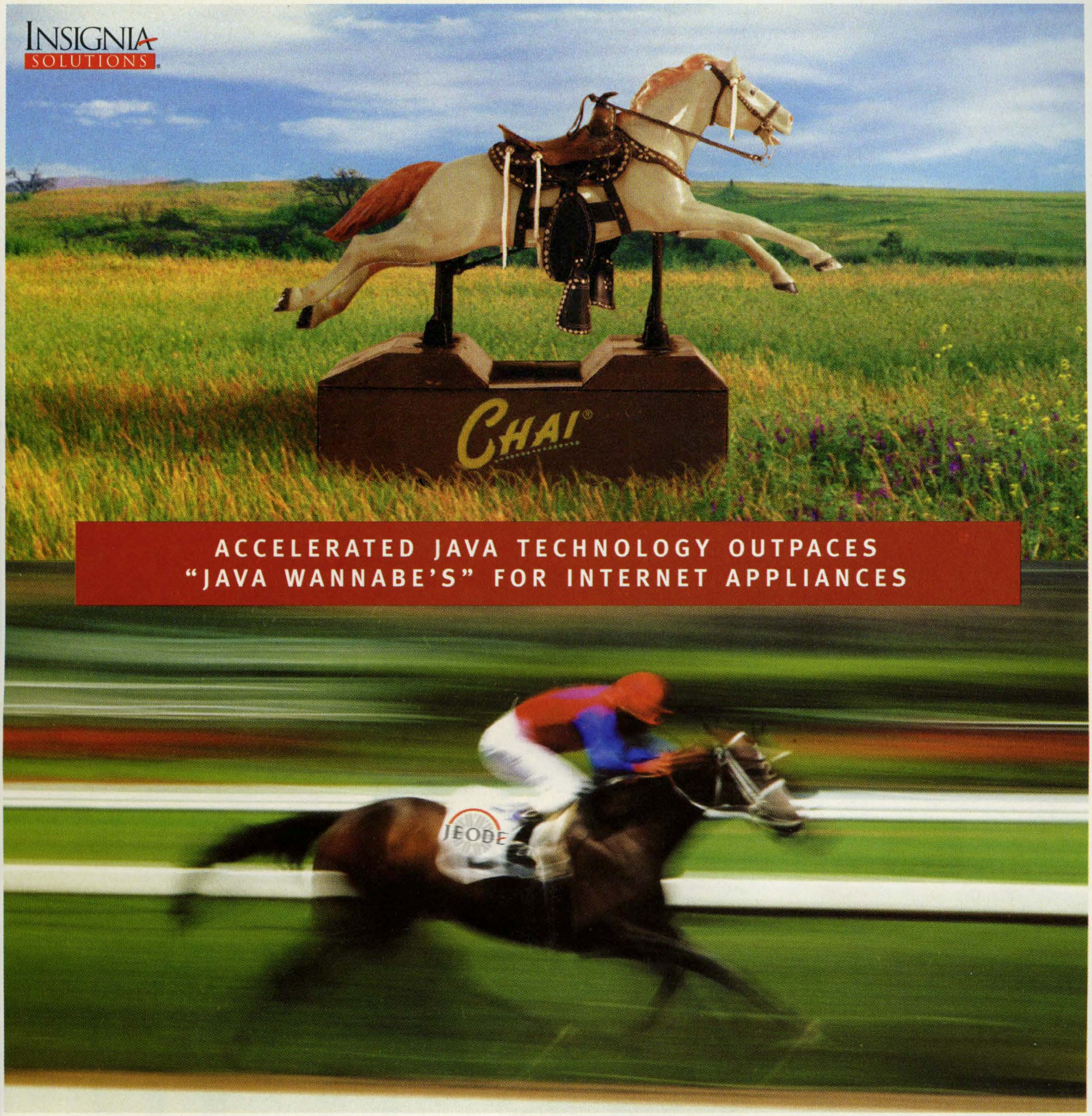
The solution in this case was to use a different representation for the parameter data, in which parameter values were stored directly in named fields of the managed objects. Any encoding and decoding necessary for interfacing to the higher-level configuration management code is now done in simple accessor (get and set) methods for the fields of interest. A utility such as `javah` can be used to generate a C header file showing the layout of the managed object as a `struct`; C code with access to this `struct` definition can then read the parameter fields directly in their native format.

As embedded devices become more complex and more networked, object-oriented development and Java will continue to become more appealing for the developers who must use and maintain them. However, efficient use of machine resources will continue to be a critical success factor for most embedded software project. Ahead-of-time compilation is one of several techniques that will permit some of us to have the best of both worlds. **esp**

Greg Wickham is a software engineer at Calix Networks. He has been developing both object-oriented and embedded applications for the past 10 years. Greg holds a BS in engineering physics from the University of Colorado at Boulder. He can be reached at greg.wickham@calix-networks.com.

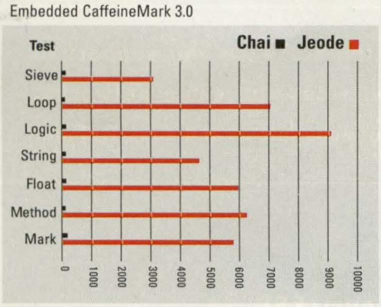
References

1. As measured by the fraction of total CPU time spent allocating and deallocating memory.
2. *www.rtj.org*



ACCELERATED JAVA TECHNOLOGY OUTPACES "JAVA WANNABE'S" FOR INTERNET APPLIANCES

In the race for high performance Java™ solutions for Internet appliances and embedded devices, Java technology plus acceleration is the winner. Jeode™ is Insignia's accelerated, fully compatible and certified implementation of the PersonalJava™ and EmbeddedJava™ specifications. It leaves non-certified Chai® six lengths back in head-to-head tests (see chart). How? At Jeode's core is the fully compatible, Embedded Virtual Machine™ (EVM™), an accelerated runtime engine that delivers high performance through patented Dynamic Compilation. The EVM is robust, has a small memory footprint, and it's highly configurable. So get Jeode for accelerated Java technology. It's real. It's proven. And it's blazingly fast. Win big!



www.insignia.com/jeode or call **1.800.848-7677.**

JAVA
Sun Authorized Virtual Machine

JEODE™

Accelerated Java Solutions
for Internet Appliances and
Embedded Devices.

Insignia and Insignia Solutions are registered trademarks and Jeode, Embedded Virtual Machine, and EVM are trademarks of Insignia Solutions, Inc. Sun, Java, EmbeddedJava and PersonalJava are trademarks of Sun Microsystems, Inc. CaffeineMark is a trademark of Pendragon Software. Chai is a registered trademark of Hewlett-Packard Company.

Convergence. It's happening. Now. How do you get there? Trillium. We provide the software and support to enable you to develop equipment for the convergence of telephony, Internet, wireless and cable networks. Call us at +1-310-442-9222 or visit www.trillium.com. Now. Because the future is ahead of schedule ... and so is Trillium.

TRILLIUM®

The World's Source for Communications Software™
Trillium Digital Systems, Inc. Los Angeles, California

www.trillium.com/ESP

CHARLES KNUTSON

Infrared Communications with IrDA

The IrDA data communication standard allows embedded systems to "speak" to one another and cooperate. Key features make operation simple. This article explores the required and optional protocols and possible implementation strategies.

Since infrared data communications, based on standards from the Infrared Data Association (IrDA), are widely available on personal computers and peripherals, it makes sense to add the same sort of short-range wireless communications to certain types of embedded systems. The IrDA standards were developed rapidly (compared to most standards organizations), and information on the IrDA protocols has not yet reached every corner of the embedded systems universe. This article gives an overview of the IrDA protocols with specific comments on their use in embedded environments.

The Infrared Data Association is an industry-based group of over 160 companies that have developed communication standards especially suited for low-cost, short-range, cross-platform, point-to-point communications at a wide range of speeds. These standards have been implemented on various computer platforms and, more recently, have become available for many embedded applications.

The protocol stack

Communications protocols deal with many issues, and so are generally broken into layers, each of which deals with a manageable set of responsibilities and supplies needed capabilities to the layers above and below. When you place the layers on top of each other, you get what is called a protocol stack, rather like a stack of pancakes or a stack of plates. An IrDA protocol stack is the layered set of protocols particularly aimed at point-to-point infrared communications and the applications typical of that environment.

Communications protocols ... are generally broken into layers, each of which deals with a manageable set of responsibilities and supplies needed capabilities to the layers above and below.

FIGURE 1 IrDA protocol layers

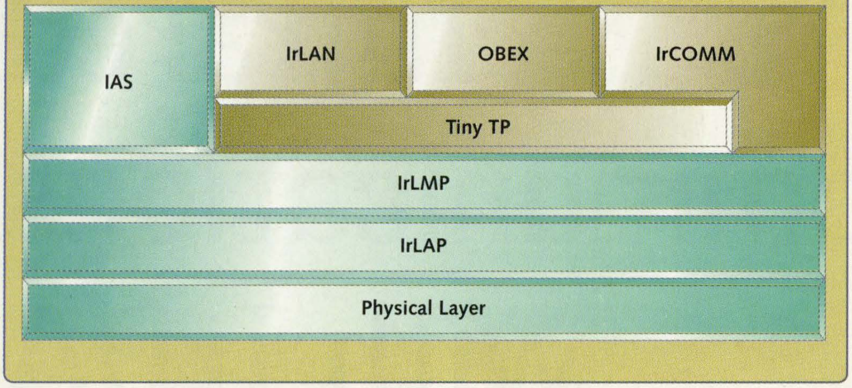


FIGURE 2 IrDA protocol layers integrated

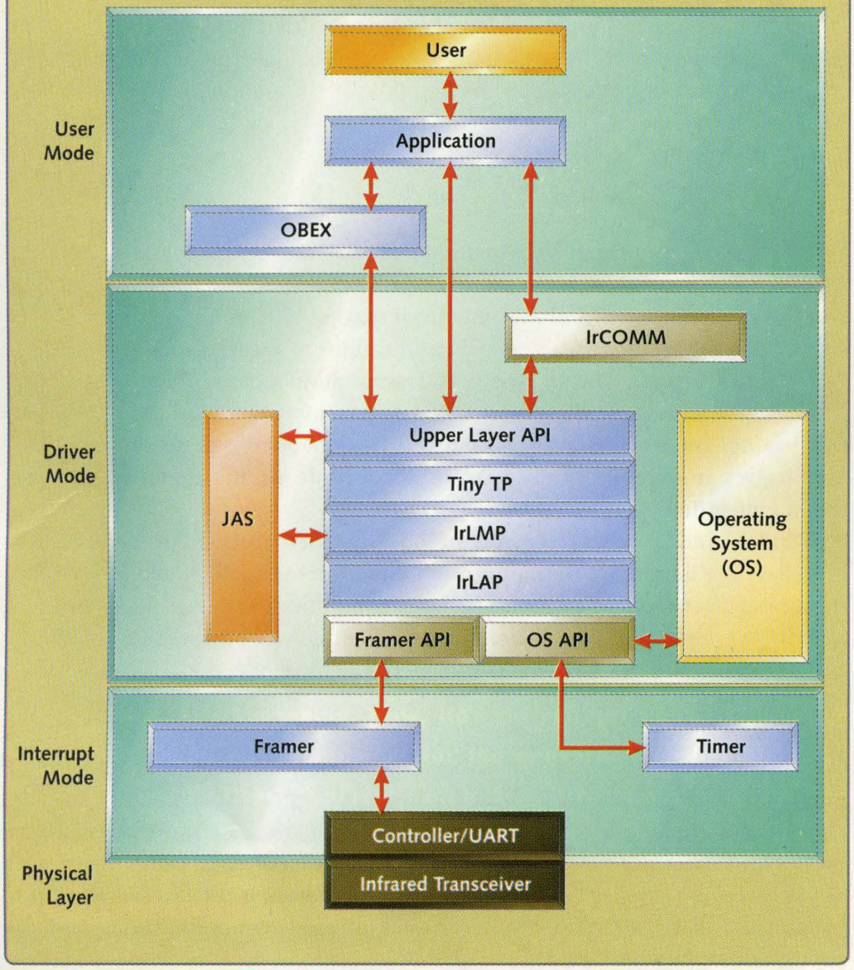


Figure 1 shows the IrDA protocol layers. This layering will serve as the overall structure for much of the remaining discussion. The layers within this stack can be divided into two groups—required and optional protocols.

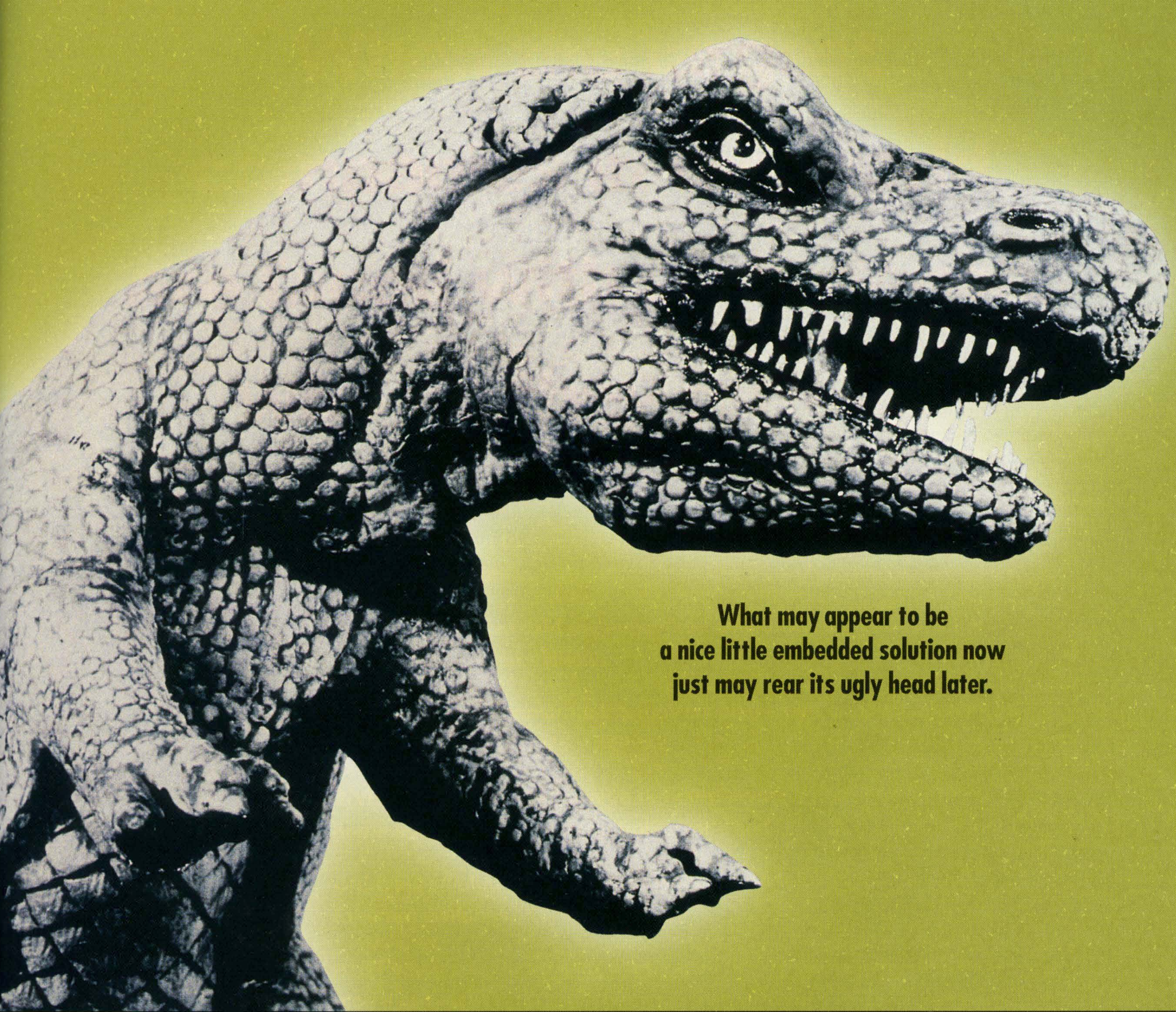
Required protocols. The required layers of an IrDA protocol stack are in unshaded boxes in Figure 1 and include the following:

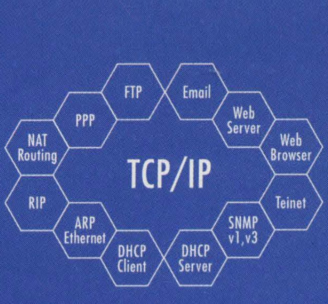
- Physical layer: specifies optical characteristics, encoding of data, and framing for various speeds
- IrLAP (Link Access Protocol): establishes the basic reliable connection
- IrLMP (Link Management Protocol): multiplexes services and applications on the LAP connection
- IAS (Information Access Service): provides a "yellow pages" of services on a device

Optional protocols. The optional protocols are shown in shaded boxes in Figure 1. The use of the optional layers depends upon the particular application. The optional protocols are:

- TinyTP (Tiny Transport Protocol): adds per-channel flow control to keep things moving smoothly. This is a very important function and is required for certain uses
- IrOBEX (Object Exchange Protocol): allows for the easy transfer of files and other data objects
- IrCOMM: provides serial and parallel port emulation, enabling existing apps that use serial and parallel communications to use IR without change
- IrLAN (Local Area Network access): enables walk-up IR LAN access for laptops and other devices

When the stack layers shown in Figure 1 are integrated into an embedded system, the picture may look more like Figure 2.


What may appear to be
a nice little embedded solution now
just may rear its ugly head later.


FTP
Email
PPP
Web Server
NAT Routing
Web Browser
TCP/IP
RIP
Telnet
ARP Ethernet
SNMP v1,v3
DHCP Client
DHCP Server

[The framer's] primary responsibility is to accept incoming frames from the hardware and present them to the IrLAP.

Physical layer

The physical layer includes the optical transceiver, and deals with shaping and other characteristics of infrared signals including the encoding of data bits, and some framing data such as begin and end of frame flags (BOFs and EOFs) and cyclic redundancy checks (CRCs). This layer must be at least partially implemented in hardware, but in some cases is handled entirely by hardware.

In order to isolate the remainder of the stack from the ever-changing hardware layer, a software layer called the *framer* is created. Its primary responsibility is to accept incoming frames from the hardware and present them to the IrLAP. This includes accepting outgoing frames and doing whatever is necessary to send them. In addition, the framer is responsible for changing hardware speeds at the bidding of the IrLAP layer, using whatever magic incantations the hardware designer invented for that purpose (these signals have not yet been standardized).

Link access

Immediately above the framer we encounter the IrLAP layer, also known as the Link Access Protocol. IrLAP is a required IrDA protocol corresponding to OSI layer 2 (data link protocol). It is based on High-Level Data Link Control (HDLC) and Synchronous Data Link Control (SDLC), with extensions for some unique characteristics of infrared communications.

IrLAP provides reliable data transfer using the following mechanisms:

- Retransmission
- Low-level flow control. (TinyTP provides high-level flow control and should almost always be used in place of IrLAP flow control.)
- Error detection

By dealing with reliable data transfer at a low level, upper layers are free from this concern and can be assured that their data will be delivered (or at least that they will be informed if it was not). Data delivery might fail if the beam path was blocked. For instance, someone could put a coffee cup in the path of the infrared beam. IrLAP alerts the upper layer so that higher-level layers can deal with the problem appropriately. As an example, an application sitting on the stack could be alerted of an interruption in data flow, allowing it to alert the user through some interface. The user could then potentially remedy the problem (by moving the coffee cup) without dropping the connection or losing the data transferred to that point.

Several environmental factors influenced the development of the IrLAP layer. These include the following:

- Point-to-point. Connections are one-to-one, such as camera to PC or data collector to printer. The range is typically zero to one meter, although extended range up to 10m or more is under development. This is not like a local area network (many-to-many) protocol
- Half-duplex. Infrared light, and therefore data, is sent in one direction at a time. However, the link changes directions frequently and can simulate full duplex in cases where timing is not extremely sensitive
- Narrow infrared cone. The infrared transmission is directional within a 15-degree half-angle in order to minimize interference with surrounding devices
- Hidden nodes. Other IR devices approaching an existing connection may not be immediately aware of the connection if they approach from behind the current transmitter. They must wait and see if the link turns around before stepping in
- Interference. IrLAP must overcome interference from fluorescent lights, other IR devices, sunlight, moonbeams, and so forth
- No collision detection. The design of the hardware is such that collisions are not detected, so the software must handle cases where collisions cause lost data with methods such as random back off

The two parties to a LAP connection have a master-slave relationship with differing responsibilities (and resulting code complexity). The IrDA terms for this are primary (master) and secondary (slave).

Primary station:

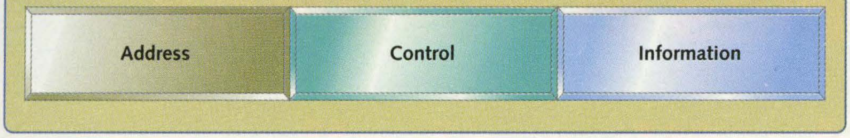
- Sends Command frames—initiates connections and transfers
- Responsible for organization and control of data flow
- Deals with unrecoverable data link errors
- Typical primary devices include PCs, PDAs, cameras, and anything that needs to print (printers are currently all secondaries)

Secondary station:

- Sends Response frames—only speaks when spoken to
- Typical secondary devices are printers and other peripherals, and resource-constrained devices (secondaries are smaller and less complex)

In any connection one device must play the primary role. The other device must play the secondary role, but its protocol stack may be either a secondary or another primary—most primaries can play a secondary role. Once started, the two sides take turns talking with the primary leading off. No side can talk for more than 500ms at a time before allowing the other side a chance to talk (even if just to say it has nothing to send for the moment).

FIGURE 3 The basic IrLAP frame format



Note that the issue of master vs. slave becomes much less obvious at the higher protocol layers—*once two devices are connected,* an application on a secondary (slave) can initiate an operation just as easily as an application on the primary side.

IrLAP is built around two modes of operation, corresponding to whether or not a connection exists.

Normal disconnect mode. Normal disconnect mode (NDM) is also known as contention state, and is the default state of disconnected devices. In this mode a device must observe a set of media access rules. Of utmost importance, a device in NDM must check whether other transmissions are occurring (a condition known as media busy) before transmitting. This is accomplished by listening for activity. If no activity is detected for greater than 500ms (the maximum time for the link to turn around), the media is considered to be available for establishment of a connection.

A great ease-of-use feature is provided by the NDM communications rules. A classic problem is getting both sides of the link configured with the same communications parameters—frequently users get completely stuck. This can be particularly difficult on embedded devices that don't have a user interface for setting or reviewing communications parameters. This problem is completely absent with IrDA solutions—all NDM communications use the following link parameters: ASYNC, 9600bps, eight bits, no parity. During the connection process, the two sides exchange capability information, and subsequently shift to the best parameters supportable by both sides.

Normal response mode. Normal Response Mode (NRM) is the mode of operation for connected devices. Once both sides are talking using the best possible communication parameters (established during NDM), higher stack layers use normal command and response frames to exchange information.

IrLAP frame format and services

The basic IrLAP frame format is shown in Figure 3. While there are too many details about frame formats to discuss here, it is worth noting that the Address and Control fields require only two bytes total—the IrDA protocols add very little overhead to the user data.

Before sending, a frame is wrapped with framing information. Three different frame wrappers are used by IrLAP, depending upon the speed of the connection.

- Asynchronous (ASYNC) framing: 9600bps to 115.2kbps
- Synchronous (SYNC) HDLC framing: 576kbps and 1.152Mbps
- Synchronous 4 PPM framing: 4Mbps

IrLAP operations are described in the specification using service primitives. You can think of the service primitive as a *conceptual model* of an API for an operation that IrLAP performs (actual APIs for IrLAP services are completely up to the developer). Figure 4 illustrates an operation: it starts with a service *request,* travels across the link as a frame, is reported as an *indication* (frequently an up-call) on the receiving side; the receiver then formulates a *response* which travels back as a frame, finally resulting in a *confirm* (often an up-call) to the original requester.

A number of services are defined in the IrLAP specification. Not all services are necessary for all devices, and the IrLAP specification (along with the IrDA Lite standard) describes the minimum requirements. The most important services include the following:

- Device Discovery: Explores the nearby IR-space to see who is present and get some hint as to what they can do
- Connect: Chooses a specific partner, negotiates the best possible communication parameters supported by both sides, and connects
- Send Data: The whole reason for all this effort—used by higher layer protocols for almost all of their work
- Disconnect: Closes down and returns to the NDM state, ready for a new connection

Link management

The IrLMP layer depends on the reliable connection and negotiated performance provided by the IrLAP layer. IrLMP is a required IrDA layer, and provides the following functionality:

- Multiplexing: LMP allows multiple IrLMP clients to run over a single IrLAP link
- Higher level discovery, consisting of:
    - Address conflict resolution on IrLAP Discovery. Handles the case of multiple devices with the same IrLAP address by telling them to generate new addresses
    - Information Access Service (IAS). A "yellow pages" describing the services available on a device

In order to have multiple IrLMP connections on a single IrLAP connection, there must be some higher level addressing scheme. The following terminology is used to describe this addressing:

Windows CE is connecting people.

VISIT OUR WEBSITE FOR YOUR EVALUATION COPY OF
WINDOWS CE PLATFORM BUILDER 3.0
ACT NOW!

Navic
systems

Imagine a payphone that merges voice and data—transforming an ordinary appliance into a personal communications portal where callers pay for transactions, change hotel reservations, or purchase tickets. A public phone terminal that provides detailed information about local restaurants, events and weather; sports results; stock performance and information, or bus schedules. Elcotel imagined and then built the Grapevine terminal, with the help of Navic Systems and the Microsoft® Windows® CE operating system.

Windows CE offers a wide array of connectivity options, including TCP/IP, WinSock, IP Multicast, Flash, IrDA, and irSock. Which enables Elcotel to connect callers with the information they need. And with support for SSL 2.0 and 3.0, Windows CE helps secure financial transactions, so callers can make instant purchases safely.

Windows CE is modular so you can customize features like sound, I/O, device drivers, and file systems. That lets Elcotel scale down code for future standalone units or reuse the modules for other devices. And Elcotel can add more features as quickly as they dream them up.

Win the race to market. Power your next generation of intelligent devices with Windows CE.

Visit www.microsoft.com/windowsce/embedded

Where do you want to go today?® **Microsoft®**

Given the limited number of LSAP-SEL values, services are not assigned fixed "port addresses" as in TCP/IP.

- LSAP (Logical Service Access Point): the point of access to a service or application within IrLMP (for example, a printing service). It is referenced with a simple one-byte number, the LSAP-SEL (described next)
- LSAP-SEL (LSAP Selector): a one-byte number that corresponds to an LSAP. Think of this as the address of a service within the LMP multiplexor. This byte is broken into ranges—0x00 is the IAS server, 0x01 through 0x6F are legal LMP connections, 0x70 are for connectionless services (not discussed in this article), and the rest are reserved for future use

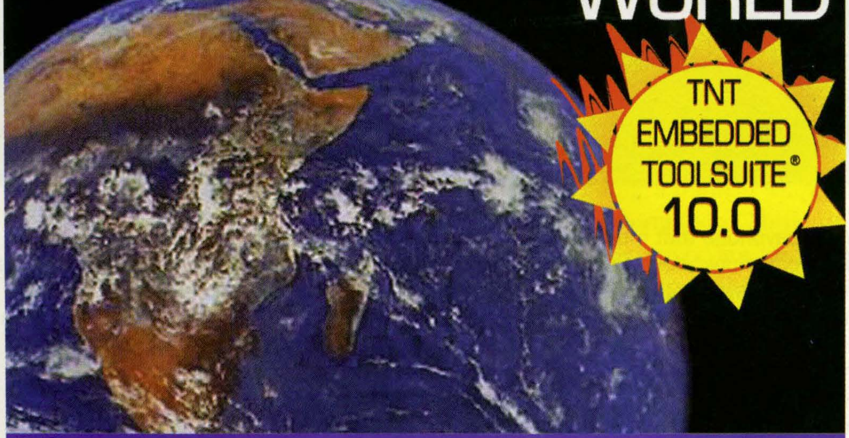
Given the limited number of LSAP-SEL values, services are not assigned fixed "port addresses" as in TCP/IP. Instead, services have fixed published names, and the LMP IAS (yellow pages) is used to look up the LSAP-SEL for a desired service.

Here are the services defined in the IrLMP specification. Not all services are necessary in all devices, and the specification (along with the IrDA Lite standard) describes the minimum requirements. Notice that this set is identical to the set listed in the IrLAP section above—it is a common feature of protocol stacks for operations to propagate upward like this, with each layer adding its particular contribution.

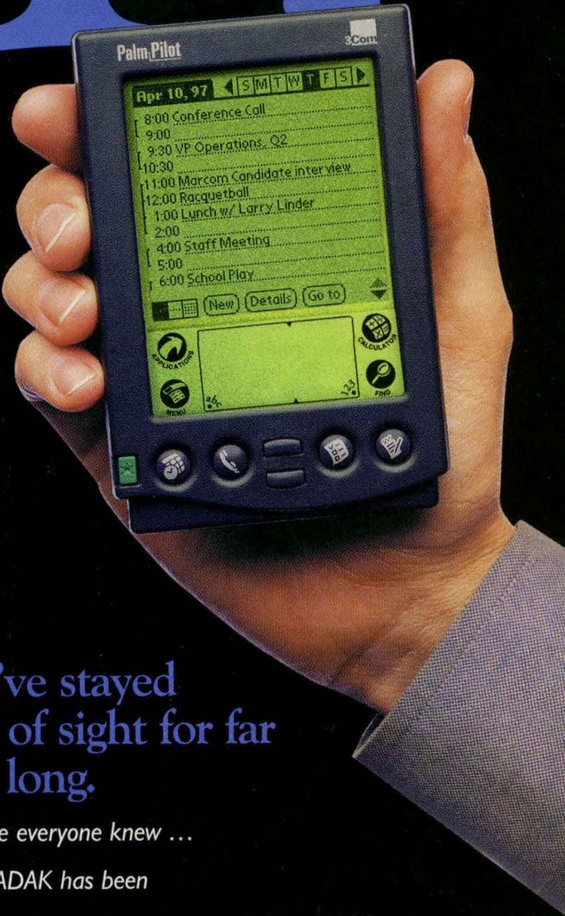
- Device Discovery: finds out additional information about devices in the IR space
- Connect: establishes a connection between a pair of services at the LMP level
- Data: sends data back and forth
- Disconnect: closes this LMP connection. Note that this does not necessarily close the LAP connection—other LMP connections may still be open

The IrLMP layer adds the two bytes of information shown in Figure 5 to frames in order to perform its basic operations:

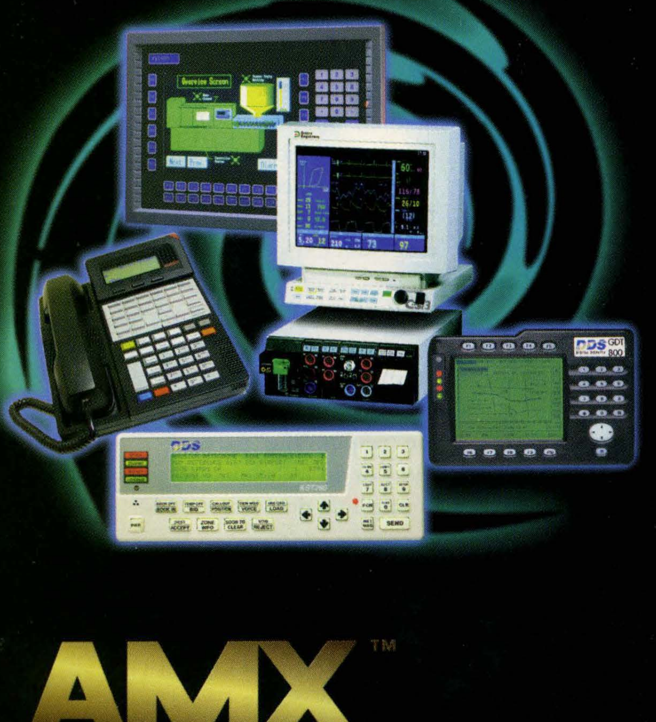
- C: Distinguishes between control and data frames
- r: Reserved
- DLSAP-SEL: LSAP-SEL (service address) of the destination of the



HEY!



We're in here!



We've stayed out of sight for far too long.

It's time everyone knew ...

that KADAK has been

quietly setting real-time standards since 1978.

That's right, from Sony to Hewlett Packard, Philips to Hughes Aircraft, over 1,000 of the most demanding companies in the world have chosen our AMX™ kernels to provide the processing power for all manner of smart devices.

From hidden embedded devices to the market-leading 3Com PalmPilot™ connected organizer, they've come to count on KADAK for superior real-time multitasking kernels, crystal-clear documentation, highly-responsive technical support and the ultimate in work and time-saving tools and utilities.

So, next time you need a real-time multitasking kernel remember ... You can count on KADAK!

AMX™

Real-Time Multitasking Kernels

KADAK

Visit us at www.kadak.com

Celebrating our 20th Anniversary

KADAK Products Ltd. 206-1847 W. Broadway, Vancouver, BC, CANADA V6J 1Y5 Tel: (604) 734-2796 Fax: (604) 734-8114 E-mail: amxsales@kadak.com

All of the services/applications available for incoming connections must have entries in the IAS which can be used to determine the service address (LSAP-SEL).

FIGURE 4 An IrLAP operation

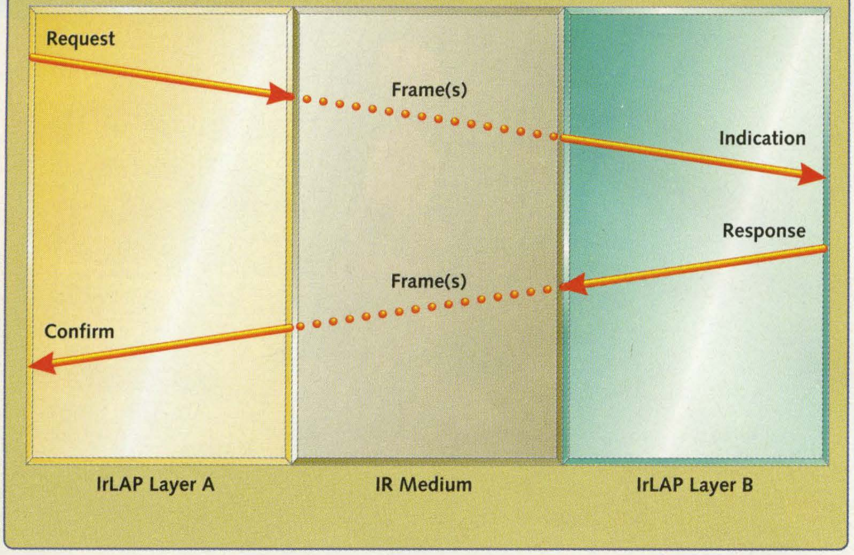


FIGURE 5 Technical art specs

7	6	5	4	3	2	1	7	6	5	4	3	2	1
C	DLSAP-SEL						r	SLSAP-SEL					

current frame

- SLSAP-SEL: LSAP-SEL for the sender of the current frame

Information access

The IAS, or Information Access Service, acts as the "yellow pages" for a device. All of the services/applications available for incoming connections must have entries in the IAS which can be used to determine the service address (LSAP-SEL). The IAS can also be queried for additional information about services.

A full IAS implementation consists of client and server components. The client is the component that makes inquiries about services on the other device using the Information Access Protocol (IAP, used only within the IAS). The server is the component that knows how to respond to inquiries from an IAS client. The server uses an information base of objects supplied by the local services/applications. In fixed-purpose embedded systems this may be a hard-coded collection of objects, while a PDA may have APIs for registering and de-registering services. Note that devices which never initiate LMP connections might include an IAS server only (no client).

The IAS Information Base is a collection of objects that describe the services available for incoming connections. The information base is used by the IAS server to respond to incoming IAS queries.

Information base objects consist of a class name and one or more attributes. They are quite similar to entries in the yellow pages of a phone book. The class name is equivalent to the business name in the phone book—it is the official published name of the service or application. IAS clients will inquire about a service using this name. The attributes contain information analogous to phone number, address, and other characteristics of the business. The one essential attribute for every entry is the LSAP-SEL (or service address), which is required in order to make a LMP connection to the service.

An IAS object is made of the following pieces:

- Class Name (up to 60 octets)
- Named attributes (up to 60 octet names):
  - Up to 256 attributes
  - Attributes value types:
    - User string (up to 256 octets)
    - Octet sequence (up to 1024 octets)
    - Signed integer (32-bit)

A number of IAS operations are defined in the IrLMP standard, but the most used, and only absolutely required, operation is `GetValueByClass`. The inquiring party gives the class name (for example, "Printer") and the name of the attribute it wants (for example, the LSAP-SEL), and receives a response that consists of one or more answers (for example, the LSAP-SELs for any printing services in the responding party's information base) or an indication that the service or attribute does not exist.

IAS Query Arguments:

- Class Name Length (1 octet)
- Class Name (Length octets)
- Attribute Name Length (1 octet)
- Attribute Name (Length octets)

Results:

- Return Code:
  - 0: Success, results follow
  - 1: No such class, no results follow
  - 2: No such attribute, no results follow

THREADX

Being Royalty Free Is not Enough.

You work under a lot of pressure to meet production demands and schedules. So the RTOS you choose has to have more than one outstanding quality to make you feel comfortable.

ThreadX, the leading royalty-free, source-code RTOS gives you more, much more...Like state-of-the-art technology with preemption-threshold and a picokernel design that automatically scales to your needs. Processor support for most processor families. Documentation so clear and concise that ThreadX's User Guide constantly receives kudos from our customers. And such a wide array of tools that chances are there is ThreadX-aware debugging already built-into your favorite tool chain.

Call **888.THREADX** or visit **www.threadx.com** and find out more about ThreadX, the RTOS with all the features to make you comfortable.

FREE
Evacuation Kit™
for pSOS users
Contact Us
Today

Processors Supported

- Win32 NEW!
- ARC
- ARM
- StrongARM
- Thumb
- PowerPC
- ColdFire/68K
- x86
- Hitachi SH
- MIPS
- NEC V8xx
- TinyJ
- SPARC
- M-Core
- TriCore
- SHARC
- TMS320C6x
- TMS320C54x

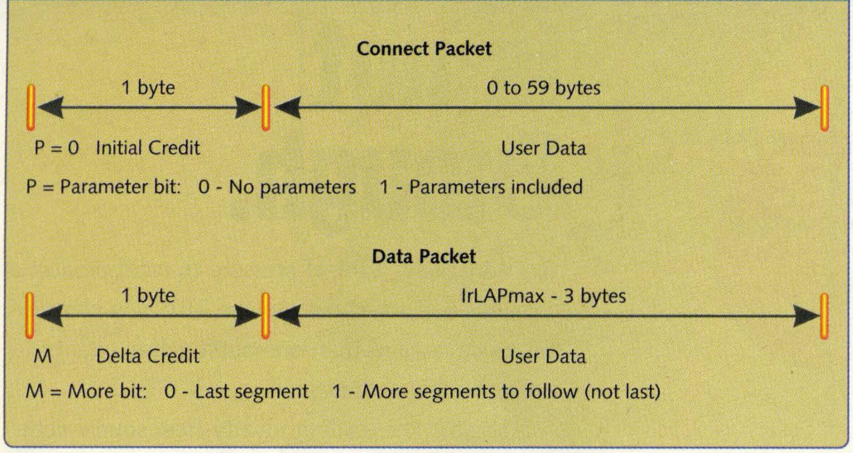
the comfortable RTOS

eL
Express Logic, Inc.

tf 888.THREADX • *fx* 858.613.6646 • www.expresslogic.com

TinyTP is an optional IrDA layer, although it is so important that it should generally be considered a required layer (except in the case of current printing solutions).

FIGURE 6 Connect packet and data packet


Connect Packet
1 byte
0 to 59 bytes
P = 0 Initial Credit
User Data
P = Parameter bit: 0 - No parameters 1 - Parameters included
Data Packet
1 byte
IrLAPmax - 3 bytes
M Delta Credit
User Data
M = More bit: 0 - Last segment 1 - More segments to follow (not last)

If the result code indicates success, the call returns the following information:

- List Length (2 octets)
- List of results:
  - Object Identifier (2 octets)
  - Attribute value (based on attribute type)

Tiny transport

To put TinyTP in its proper perspective, it would help to review the layers covered so far:

- The *physical layer* defines the hardware requirements and low-level framing of the data
- *IrLAP* provides reliable, sequenced data, and trouble-free connection at agreed upon parameters with automatic negotiation to best common parameters
- *IrLMP* provides multiplexing of services onto the LAP connection and the IAS "yellow pages" of services available for incoming connection

TinyTP is an optional IrDA layer, although it is so important that it should generally be considered a required layer (except in the case of current printing solutions). TinyTP provides two functions:

- Flow control on a per-LMP-connection (per-channel) basis
- Segmentation and reassembly (SAR)

TinyTP adds one byte of information to each IrLMP packet to perform its task.

Per-channel flow control is currently the most important use of TinyTP. You may recall that IrLAP offers flow control and wonder why another flow control mechanism is needed. To illustrate the need, suppose that a LAP connection is established and two LMP connections are made on top of the LAP connection using the LMP multiplexing capability. If one side turns LAP flow control on, the flow of data on the LAP connection (which carries all the LMP connections) is completely cut in that direction, and the other side cannot get the data it wants until LAP flow control is turned off. The work of the second side may be seriously disrupted (especially if timers are involved).

If flow control is applied on a per-LMP-connection basis using TinyTP (or other mechanisms), then one side can stop to digest information without negatively affecting the other side.

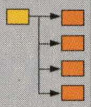
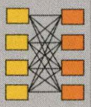
TinyTP is a credit-based flow control scheme. It works as follows:

- At connection, some credit is extended by each side. One credit corresponds to permission to send one LMP packet. If you send a credit, you must be able to accept a maximum-sized packet. You can see that the number of credits you extend depends entirely on how much buffer space you have. As long as you have buffers, you can send anywhere from one to 127 credits
- Sending data causes credit to be used up (one unit of credit per packet sent)
- Periodically, the receiver issues more credit. This "credit policy" is entirely at the receiver's discretion, but the policy can make a big difference in the performance of the link. If the sender is constantly running out of credit and having to wait for more, throughput will suffer
- If a sender has no credit, no data movement can occur, but…
- A credit-only packet can always be sent—it's not subject to flow control

Although the above description talks about the sender and receiver as if those roles are fixed, it is common for both sides of a LMP connection to send and receive, hence both sides will be issuing and using credit. Note that the credit byte normally travels as part of a LMP data packet, so LMP packets are not needed just for sending credit as long as there is data to send and credit with which to send it. Obviously a little head scratching is in order to set up an efficient credit policy.

The other TinyTP function is called segmentation and reassembly. The basic idea is that TinyTP breaks large data into pieces (segmentation), and puts it back together on the other side (re-assembly). The entire piece of

Differential Signaling made Easy.

Differential Standard	Bandwidth
LVDS Point-to-Point	36 pairs @ 622 Mbps or 344 pairs @ 311 Mbps
BLVDS Multi-Drop 	344 pairs @ 311 Mbps
Multi-Point 	344 pairs @ 311 Mbps
LVPECL	36 pairs @ 622 Mbps or 344 pairs @ 311 Mbps

High bandwidth, low power consumption and noise immunity, three device requirements necessary to ensure success in the design of today's complex electronic systems. And, to that end, more and more designers are relying on Differential Signaling to satisfy these stringent demands. That's why more and more designers have come to rely on the Virtex™-E family of FPGAs from Xilinx.

We call it "SelectI/O+™ technology". It's the freedom to interface directly to LVPECL, LVDS, and Bus LVDS devices. But it's really much more than just a simple interface. In fact,

VIRTEX™-E

through our hierarchy of differential support, you can access up to 36 LVDS and LVPECL I/O pairs at 622 Mbps, and up to 344 differential pairs at over 311 Mbps. Now that's no simple interface! In fact, the Virtex-E family can provide multiple 10 Gbps ports and still maintain extremely high signal integrity.

So, if your next design requires greater flexibility and unprecedented bandwidth, take a look at the Virtex-E family of FPGAs. Because,when it comes to Differential Signaling, the E stands for easy.

XILINX®

The Programmable Logic Company℠

www.xilinx.com/ve2.htm

The IrDA Lite specification is composed of a number of strategies. A developer may choose to implement all of them, or only the ones appropriate to a particular device.

data being chopped up and re-constituted is called an SDU, or Service Data Unit, and the maximum SDU size is negotiated when the TinyTP/LMP connection is first made.

As with the IrLAP and IrLMP layers, TinyTP operations are characterized as service primitives:

- Connect: Negotiate maximum SDU size
- Disconnect
- Data: Reliable sequenced data
- Local Flow Control: Stop data delivery
- Udata: Unreliable, unsequenced (pass through to IrLMP)

TinyTP service primitives focus on the core of the LMP primitives—connect, send, and disconnect, adding a means to exert flow control.

The two frame formats used by TinyTP are the *connect packet* (carried with the IrLMP connect packet, hence the limited data length), and the *data packet*, carried with IrLMP data packets. Both are shown in Figure 6.

IrDA Lite

Up until this section, we have discussed layers of the IrDA protocol stack. This section briefly describes a specification which does not create a new layer, but which modifies layers already described.

The IrDA Lite specification describes a set of design and implementation strategies that together yield the smallest possible implementation that will still perform connection-oriented communications with a full IrDA implementation.

Naturally, the ultimate size of the protocol stack will depend heavily on the hardware, the software tools available, and the experience and skills of the development team. However, our experience suggests that primaries under 10K code size and secondaries under 5K are feasible on CISC processors commonly used in embedded systems. RAM usage can be as low as a few hundred bytes, most of which is needed to buffer incoming frames.

The IrDA Lite specification is composed of a number of strategies. A developer may choose to implement all of them, or only the ones appropriate to a particular device. For instance, some of the strategies severely limit the performance of the stack:

- Speeds restricted to 9600bps

Pentium® Debug: $4,995

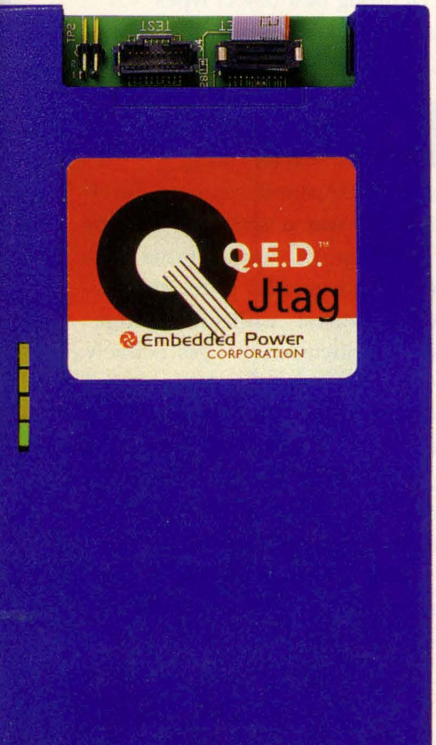
EmProve
Virtual Application Development and Prototyping (VAD-P™) environment

RTXC™
Royalty-free real-time OS
- TCP-IP stack
- File system
- Graphic system

BeaconSuite™
- Compilers
- Linkers & Locators
- Debuggers
- Simulators

Q.E.D. ™ Jtag

Sometimes the shortest distance between two points covers new ground.



Embedded Power's Q.E.D. tradition continues - with a full JTAG debug port implementation! We bring you accurate, dependable hardware-assisted source-level JTAG debug on the Windows® NT/95/98 platforms.

Using our own VisualProbe™ or other debuggers, Q.E.D. Jtag provides:

- Kernel-aware debug with our RTXC RTOS
- Debug for all Pentium family members
- Full visibility of all internals, including hidden registers
- Support for your favorite toolchains, including CAD-UL, Borland, MetaWare® and Microsoft®
- Fast, efficient host communications. High throughput means fast debug functions and quick code downloads - even to FLASH.
- Requires NO fumbling with PC bus cards! Standard connection to your PC is via an IEEE 1284/EPP parallel port.
- Eliminates the need to CAD a JTAG port to your board. We offer a complete set of JTAG breakout adapters that make hardware connection a snap!

Stay on course through Pentium-class embedded development with Q.E.D. Jtag Debugger.

Pentium Processors	JTAG Port Pinout	Breakout Board
Pentium I	20	PGA 320
Pentium II	30	PGA 370/SLOT 1
Mobile Pentium II	30	BGA/Micro BGA Intel breakout board
Pentium II Xeon	30	PGA 370/SLOT 1
Celeron	30	PGA 370/SLOT 1

www.embeddedpower.com

Embedded Power
CORPORATION

U.S. 281-561-9990
email: info@embeddedpower.com
Europe: +44 (0) 1256 474448
email: eurosales@embeddedpower.com
© 2000 Embedded Power Corporation
Pentium® is a trademark of Intel Corporation

When the IrDA standards were developed, there was a strong desire to allow existing PC applications that use serial and parallel ports to operate via infrared without change.

- LAP packet size restricted to 64 bytes

While the most severely constrained devices (watches, for instance) may both need and tolerate these constraints, many devices (such as digital cameras) need high performance. Some strategies do not affect performance at all, so a judicious choice is in order to balance the needs for performance, functionality, and size.

While a description of most of the IrDA Lite strategies is beyond the scope of this article, it is worth noting the following key strategy and its effect on the decision to use an IrDA Lite-based protocol implementation.

In a non-IrDA Lite setting, applications issue an IrLMP disconnect when they are done, and IrLMP closes the LAP connection when all the LMP connections are done. In contrast, under IrDA Lite, applications use the IrLAP disconnect operation when they are done. Suppose we have two applications communicating on their respective LMP channels with counterparts on another device. If one side finishes up and issues an IrLAP disconnect, the other side's connection will terminate without warning, a potentially disastrous scenario.

Many embedded systems have a focused purpose and will find a single communication channel adequate. If multiple applications desire separate channels *at the same time*, things will still work as long as they are aware of each other and employ a "last one to leave turns out the light" approach to the IrLAP disconnect. Alternatively, it may be much easier for all applications to send all their data through an IrOBEX implementation (discussed in the next section), and let IrOBEX take care of making and breaking connections for everyone, as well as handling all the intricacies and contingencies that occur in any communications task.

Object exchange

IrOBEX is an optional application layer protocol designed to enable systems of all sizes and types to exchange a wide variety of data and commands in a resource-sensitive standardized fashion. It addresses one of the most common applications on either PCs or embedded systems: take an arbitrary data object (a file, for instance), and send it to whomever the infrared device is pointing to. It also provides some tools to enable the object to be recognized and handled intelligently on the receiving side.

The potential range of objects is wide, encompassing not only traditional files, but also pages, phone messages, digital images, electronic business cards, database records, hand-held instrument results, or diagnostics and programming. The common thread is that the application doesn't need or want to get involved in managing connections or dealing with the communications process at all. Just take the object and ship it to the other side with the least fuss possible. It is very similar to the role that HTTP serves in the Internet protocol suite, although HTTP is very "pull"-oriented in its fundamental design, while OBEX is more evenly balanced.

OBEX was created to "package" an IrDA communications transaction as completely as possible and thereby dramatically simplify the development of communications-enabled applications. It was further designed to meet the following criteria:

- Simple: Supports most-needed operations/applications
- Compact: Under 1K code on small system
- Flexible: Supports data handling for both industry standard and custom types
- Extensible and debuggable
- Works on IrDA, but is transport independent

The OBEX standard consists of the following pieces:

- Session model: the rules of conversation governing the exchange of objects. Includes optional negotiation during connection, a set of operations such as Put and Get. Allows terminating the transfer of an object without closing the connection. Supports graceful close of a connection
- Object model: provides a flexible and extensible representation for objects and information describing the object
- Guidelines for use and extension:
  - Defining new session operations
  - Defining new object types
- IAS entry for a default OBEX server, and suggestions for its capability

Serial and parallel port emulation

When the IrDA standards were developed, there was a strong desire to allow existing PC applications that use serial and parallel ports to operate via infrared without change. These applications, collectively known as "legacy applications," included printing, file transfer applications such as LapLink or Carbon Copy, and modem communications.

However, IrDA infrared communications differs significantly from serial and parallel communications. For instance, both serial and parallel cables have individual circuits over which signals can be sent independently and concurrently. By contrast, infrared has a single beam of light, and all information must be fitted into LMP or higher layer packets in a serial stream.

The IrCOMM standard was developed to solve these problems and

allow legacy applications to be used over infrared with a minimum of hassle. The key feature of IrCOMM is the definition of a so-called *control channel* to carry the non-data circuit information. In the stack picture, IrCOMM rests on top of IrLMP and TinyTP.

IrCOMM is an optional IrDA protocol that applies only to certain applications. In general, new applications are better served if they avoid IrCOMM and use other IrDA applications protocols such as IrOBEX, IrLAN, or TinyTP directly. This is because IrCOMM masks some of the useful features built into the lower-layer protocols. After all, its job is to make IrDA look like serial and parallel media that do not have handy features like automatic negotiation of best common parameters and a "yellow pages" of available services.

Because different applications use the non-data circuits of serial and parallel communications to varying degrees, four service types are defined in IrCOMM:

- 3-Wire Raw (parallel and serial emulation): sends data only, no non-data circuit information and hence no control channel. Runs directly on IrLMP
- 3-Wire (parallel and serial emulation): minimal use of control channel. Uses TinyTP
- 9-Wire (serial emulation only): uses control channel for status of standard RS-232 non-data circuits. Uses TinyTP
- Centronics (parallel emulation only): uses control channel for status of Centronics non-data circuits. Uses TinyTP

LAN access

The final optional protocol discussed is IrLAN. It is mentioned only briefly here because it is not an approved standard at this time, nor is its use widespread in the world of embedded systems. It primarily serves as an extremely convenient connection between portable PCs and office LANs.

IrLAN offers three models of operation:

- Enable a computer to attach to a LAN via an Access Point Device (sometimes called an IR LAN Adapter). The Hewlett-Packard NetBeam IR is an example of this type of device
- Enable two computers to communicate as though attached on a LAN—in effect an instant LAN between a pair of machines, with access to the other machines' directories and other LAN capabilities
- Enable a computer to attach to a LAN through a second computer already attached

Simple communications

Features like automatic selection of compatible communication parameters and service "yellow pages" make the IrDA protocols well suited to embedded devices, even in consumer markets where the device must communicate simply to gain widespread acceptance.

The IrDA standards documents and additional information about the Infrared Data Association are available at *www.irda.org*. The IrDA Web site also includes links to suppliers of hardware and software. **esp**

Charles Knutson is an associate professor at Brigham Young University. Previously, he was vice president of research and development at Counterpoint Systems Foundry. He is also chair of the test and interoperability committeee of the Infrared Data Association. Dr. Knutson holds a PhD in computer science from Oregon State University.

Embedded Internet Tools

Linux-based software tools

White dwarf Linux (wd linux) v. 1.0 is based on the most recent Linux kernel (2.2.14) and includes more than 40 software packages, including a web browser, an Apache 1.3.11 web server, and e-mail capability. It's small enough to load on embedded PCs and contains features including TCP/IP tools (ftp, telnet, hostname, ftpd, and so on); a terminal emulation package; gcc development tools; and ppp drivers and scripts. wd linux runs on JUMPtec single board computers including the DIMM-PC, the MOPSlcd4, and the MOPS586. It consists of a 1.44MB bootdisk and a 1.44 MB rootdisk. White dwarf linux is available now through the company's Web site.

EMJ Embedded Systems
Apex, NC
(800) 548-2319
www.emj.com

Software supports encryption

The EmWeb/SSL software helps integrate Web management and encryption software by supporting RSA Security's RSA BSAFE SSL-C encryption software. It supports Secure Socket Layer (SSL) versions 2 and 3 and the Internet Engineering Task Force's Transport Layer Security (TLS) v. 1 standard for users of the VxWorks RTOS.

Agranat Systems
Maynard, MA
(978) 461-0888
www.agranat.com

Tools support Linux

Embedded PERC v. 2.2 supports the Linux operating system and includes the PERC Java-compliant virtual machine, compilers, tools, and libraries. In addition to supporting Linux, Embedded PERC also supports VxWorks, pSOSystem, Windows NT, ETS, and RTX. The Embedded PERC v. 2.2 product family also includes a set of development and integration tools, including the ROMizer static linking tool, PERCH native method integration tool, QuickPERC just-in-time and flash compiler, Embedded Toolkit component library, and PERC 2.2 libraries. It's available now for licensing.

NewMonics
Ames, IA
(515) 296-0897
www.newmonics.com

Integration utility for Platform Builder

AMCTAP integration utility for Windows CE Platform Builder v. 3.0 gives application developers working in the Windows CE development environment run-time control of Motorola 8xx processors and NEC MIPS processors. AMCTAP is a hardware debugging application program interface. With this utility, any Applied Microsystems emulator or third-party emulator that conforms to Applied's API will work with any tool in the Windows CE Platform Builder Development environment. AMCTAP provides a connection to the target through a JTAG interface to CE and operates in two distinct modes: hardware debug and software monitor mode.

Applied Microsystems
Redmond, WA
(800) 426-3925
www.amc.com

Embedded device browser

Escape v. 4.0 Professional edition is an embedded device browser that delivers an Internet browsing equivalent to the desktop in less than an 800K footprint. It's a Java-based Web browser offering write once, run anywhere capability, and allows browsing for any smart device using any embedded operating system. The architecture provides support of features such as HTML 4.0, forms, frames, tables, applets, images, JavaScript 1.4, SSL v. 2 and 3, DHTML, DOM, and CSS. Escape v. 4.0 Professional edition is available now.

Espial
Ottawa, Canada
(613) 230-4770
www.espial.com

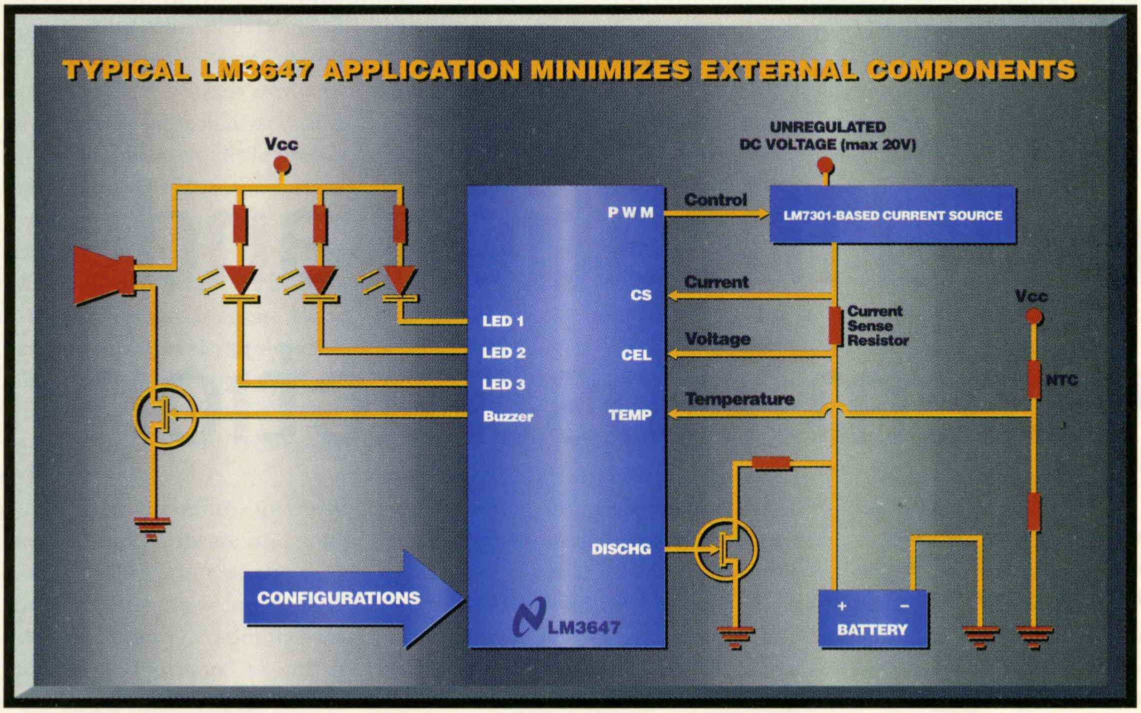
Mobile middleware

Smart IP is a mobile middleware product that allows Internet/intranet and IP-based applications to run over digital circuit-switched wireless data networks efficiently. It uses proxy server technology to support the Winsock application programming interfaced used by Internet and IP-based applications. New features include dial up networking control, auto-connect, auto-reconnet, auto-disconnect, and maintenance of persistent IP addresses. Smart IP v. 2.0.1 is now available for Windows 95, 98, and NT 3.5 or later starting at $52.50 per user.

Nettech
Princeton, NJ
(609) 734-0346
www.nettech.com

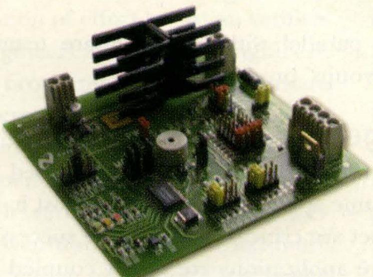
High Precision, Fast Charger Chip for NiMH, NiCd and Li-Ion Batteries

National's LM3647 12-Bit Resolution, Accurate Voltage Monitoring Within ±50mV Prevents Li-Ion Under or Over Charge



Features:

- Selectable Charge Current Reduces Charge Time to Less Than 1 Hour For All Charge Capacities

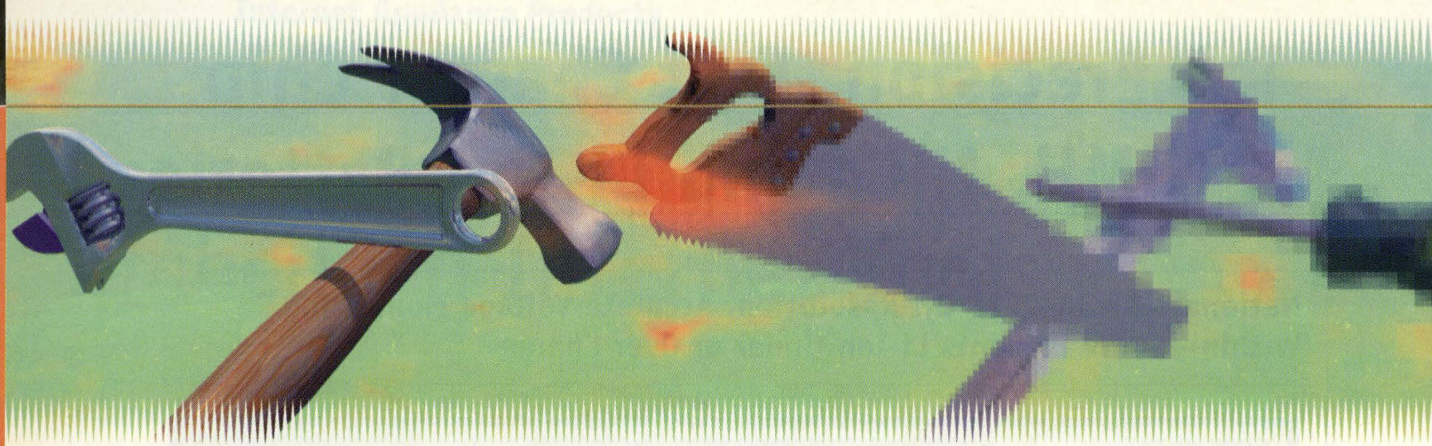


- Pre-Charge Current (Softstart) Prevents Premature Termination of Charge Process
- Batteries Left in the Charger are Always Maintained at 100% Charge
- Multiple, Selectable Fast Charge Termination Methods Provide Design Flexibility to Optimize Both Cost and Performance
- Optional Temperature Monitoring Ensures Batteries are Within Specified Temperature Limits Before Battery Charging Begins
- Dynamically Detect Battery Insertion, Removal, Short Circuit, and Bad Battery With No Extra Hardware

For More Information on LM3647: www.national.com or 1-800-272-9959
Order your LM3647 Evaluation Kit Online: http://www.national.com/pf/LM/LM3647.html

National
Semiconductor

BOB MORASSE



Co-Verification and System Abstraction Levels

System-on-chip designs require development tools and methodologies that enable concurrent design, debug, and verification of hardware and software. This article identifies key attributes of the various system abstraction levels and a methodology that describes how hardware and software designers use them to work together in a co-verification development environment.

Application-specific integrated circuits (ASICs) are generally developed by cooperating but separate teams of hardware and software engineers. In addition, a distinct team of verification engineers usually works closely with the hardware team to specify and develop test suites for the chip. Too often, the work of the software team lags, while they wait for a development platform on which to implement and test their code. This means that the first integration of hardware and software occurs only when the first silicon is available, at a time when undetected architectural flaws and various bugs in the ASIC are very costly to correct. This may force a re-design of the ASIC or a less expensive but potentially performance-degrading software fix. Ideally, the software team would be able to work in parallel with the hardware team, thus increasing both groups' productivity.

New challenges

System-on-a-chip (SOC) designs are primarily used in specialized, high-volume applications where low-cost hardware and time to market are critical factors in the success of the product. Software applications are highly coupled to custom processor and DSP-based ASIC hardware solutions. Central to the process is co-development of hardware and software. In order to get to market on time despite shrinking design schedules and the need for ASIC one-pass success, hardware and software must be developed and verified concurrently. The design process must maximize productivity by minimizing time-consuming steps along the way.

Ideally, the software team would be able to work in parallel with the hardware team, thus increasing both groups' productivity.

SOC design presents additional challenges in that there is limited visibility of the inner structure of the hardware, making silicon debugging difficult and expensive. Many of the common embedded systems development tools, such as logic analyzers and in-circuit emulators, are of little use, as there may be no access to the system bus and other signals internal to the ASIC. Often, the only way to debug the final silicon is through a JTAG or similar serial interface. In some cases even these are not provided unless the development team gets together and agrees in advance that there is a true need to bring out some additional pins for debugging.

It is desirable, therefore, to verify correctness of the hardware and software prior to fabrication of the first actual chip. This feat is accomplished via hardware/software co-simulation, logic emulation, hardware prototyping, or a combination of all three. These technologies let the design team model the hardware and software in a number of ways. Such models represent abstractions of the final system in various levels of detail.

The biggest opportunities for increased efficiency lie in eliminating duplication of effort by system verification engineers and software engineers, and by providing a means for software engineers to start their design work earlier in the project. Verification engineers spend most of the project doing hardware verification by building software-driven test benches for hardware designs under simulation. Test bench reuse is targeted to follow-on projects, if at all. Verification software is not used in the actual production system. This is a significant amount of work, to be essentially thrown away. On the other hand, software engineers will be developing low-level device drivers and system diagnostics that are designed to exercise the hardware. They traditionally get started late in the process because they are waiting for prototype hardware or actual silicon to port and debug their code.

The solution is to provide a marriage of the hardware and software development environments such that software engineers can write production code that also provides verification test vectors for the HDL-based (hardware description language, for example, Verilog or VHDL) hardware design. At the same time, simulated hardware is used to verify correct software functioning. Hardware, software, and verification engineers can work concurrently on the same team, increasing the efficiency of the design process, thus getting a higher quality product to market in less time.

Co-verification tools

Within the last few years a number of EDA companies, working with various software and embedded tools vendors, have introduced products that are designed to interface HDL-based hardware design simulations with software development and debug environments. Commercial co-verification tools include Mentor Graphics' Seamless, Synopsys' Eagle Design Tools, and V-CPU from Summit Design (now merged with Viewlogic Systems into a new company called Innoveda). There are various third party add-ons available for most of these. They support nearly all the major hardware simulators, waveform generators, and debuggers for Verilog and VHDL. Software development support is available for major providers such as Wind River Systems, Green Hills, Microtec, and Microsoft. The upside of all this is that hardware and software engineers are able to use many of their familiar tools in the ways they normally would, while not actually working with real silicon.

Although their individual features and capabilities vary, all of the co-verification tools use similar approaches to solving the problem. They all provide the following basic functionality:

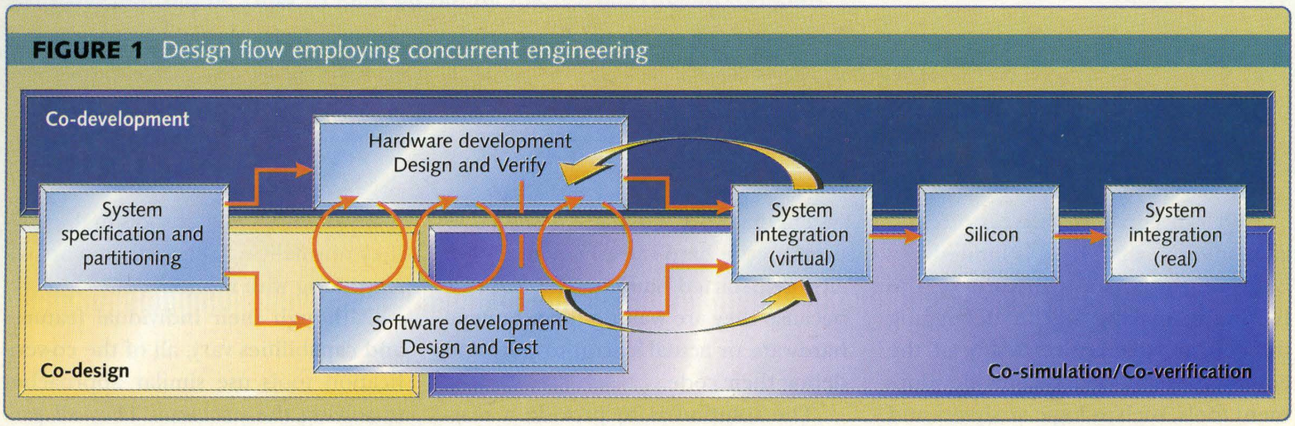
- Hardware/software simulation control, including tool configuration, simulation initialization/startup, and breakpoint synchronization
- Data and event passing between the software and hardware environments
- Support for various levels of system abstraction and software execution optimizations

Each of the tools provides a virtual control panel that is used to link and synchronize the hardware and software tool sets. Hardware models and software executables can be loaded for simulation. Breakpoints in either domain can cause both simulations to stop, allowing debug from either side. Software can access hardware for reads or writes. Hardware events, such as interrupts and resets, can be sent to the software for processing. This connection is generally a socket-based protocol that uses a C language interface at the HDL side and either a trap mechanism or tool-specific C function call on the software execution side.

Figure 1 depicts a design flow that utilizes concurrent engineering as a development methodology. It shows an iterative design and verification process in which hardware and software are designed and tested against each other on a virtual simulation platform. Note that hardware and software development is done in parallel with continuous verification along the way. Deliverables from the hardware

RUPERT ADLEY

FIGURE 1 Design flow employing concurrent engineering


Co-development
Hardware development Design and Verify
System specification and partitioning
System integration (virtual)
Silicon
System integration (real)
Software development Design and Test
Co-design
Co-simulation/Co-verification

and software teams must be coordinated such that design and verification of each hardware and software module can be built from a baseline completed in the previous stage. This will require up-front communication and planning as part of the system specifications and project scheduling phases.

System abstraction levels

Systems can be modeled with various degrees of detail that define a level of abstraction. These levels range from the complete system in silicon, which may be defined as abstraction-level zero, to a pure application level software representation that uses stubs to represent the hardware interface. Each abstraction level constitutes a unique configuration, or system view, where various design, debug, and verification activities can be done most efficiently. The design methodology defines how and when to use the various abstraction levels within the development lifecycle.

Co-verification tools support a wide range of system abstractions, interfacing to hardware emulators at the lower levels (greater detail) to C/C++-based behavioral models of hardware at the higher levels of abstraction. In most current designs, hardware is generally modeled and simulated using an HDL (typically, in Verilog or VHDL). The speed of the logic simulation is directly related to the number of hardware blocks in the model, and the level of detail of the model and thus the level of abstraction.

In some cases, the ASIC logic may be compiled for hardware emulation. FPGA devices are programmed with the ASIC circuitry and tested using traditional tools. Commercially available hardware emulators, that are configurable to your design, can be purchased from vendors such as QuickTurn (part of Cadence Design Systems) and Aptix. Alternatively, design-specific hardware prototype boards may be built in-house using the same FPGA technology. This article will discuss hardware emulation for completeness, but, due to the costs involved, logic simulation is far more common and thus more relevant to the discussion.

On the software side, three possible processor models, or levels of abstraction, are supported. These are, from the most to the least abstract:

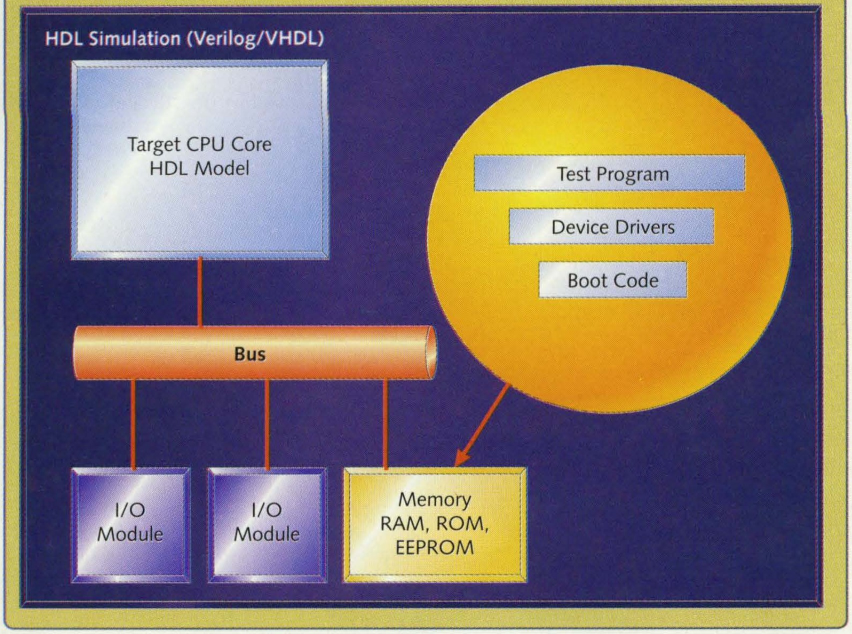
- Host compiled C or C++
- Host-based target CPU instruction set simulator (ISS) supporting C, C++, assembly language
- Emulator with actual target-based CPU supporting C, C++, and assembly language

Software execution speed is generally not the gating factor in a co-verification environment where there is a tight interface to hardware modeled with HDL, as the logic simulation will always be orders of magnitude slower, but the ISS has the lowest performance because it is a simulation itself. For computation-intensive applications, host-based C/C++ offers the best software execution performance. All the tools support host-native C/C++ and ISS. At least one tool provides an emulator interface for a number of target CPUs. ISS and bus model support for many DSP cores are in the works.

Different levels of detail are required to debug and verify the design at the module and system level at different points in the design and verification phases. A system test may be required to verify memory and peripheral bus transfer timing from a CPU or DSP under software control. This requires a detailed hardware model of the hardware surrounding the processor, possibly a gate level simulation, which would be very time consuming. Very little software can be executed due to excessive logic simulation time. Once proper bus functionality is demonstrated, a less detailed, higher level of abstraction model that optimizes software execution may be used for software design and debug. Moving instruction and data fetches from HDL memory models to the software run-time environment (host memory for native or ISS modes, actual RAM in emulator systems) significantly increase software throughput by operating at a higher level of abstraction.

Multiple models and configurations may be available for use by the project team at different times and for different purposes in the development process. Hardware-intensive verifica-

FIGURE 2 Abstraction level 2: full system HDL



tion will generally involve lower level abstractions that are simulation intensive. Software engineers will normally want to use a higher level of abstraction, minimizing hardware detail in order to boost software execution speed. Some tools provide a means for on-the-fly optimizations during interactive sessions. This feature allows the hardware or software engineer to control the level of software/hardware interaction during a debug or test session, effectively creating a dynamic abstraction level. For instance a hardware interrupt may signal a CPU that data is available for some extensive computation. It would be possible to run the session with the logic simulation in tight synchronization with the software execution until the interrupt is asserted. At that point the logic simulation can be halted and the software allowed to execute at full speed, processing the data.

The following sections describe the various system abstractions generally supported by currently available off-the-shelf tools. In some cases the design team may provide software stubs or behavioral models to support a higher-level system abstraction required for a certain design or verification task. I refer to the abstraction levels starting with level 0 (no abstraction) to level 6. This numbering scheme has no particular significance.

Level 0: Silicon. This is where the silicon ASIC first meets the board. Eventually the development team will have to integrate hardware and software on the finished part. Nothing abstract about this. If the methodology and use of tools has been successful there should be no showstoppers at this point.

Level 1: Hardware emulation. As mentioned earlier the ASIC logic can be prototyped using FPGA technology. Large, relatively fast FPGAs are now available that can hold complex logic designs. Configurable emulators from companies such as Quickturn and Aptix allow many different designs to be modeled and emulated. Once programmed, the hardware emulator can be run and tested much like any hardware prototype. Silicon processor cores can be added to the prototype and downloaded with actual code. Alternatively, HDL-based "soft cores" can be loaded directly into the FPGA array and used to execute software. The software can be executed and debugged in a similar way to the actual hardware. In most cases software will execute in real-time, allowing significant software verification including RTOS boot-up and test.

The main drawback to this approach is cost. Fully configurable hardware emulators can run many thousands of dollars. Also there may be significant extra design work to "tweak" the FPGA prototype in the synthesis phase from RTL to gates to get it to work as required. This is due to the different way FPGA logic is connected internally vs. the full-blown ASIC. The FPGA will generally not run as fast as an ASIC. FPGAs will have some important timing and signal characteristic differences with respect to the final ASIC.

Hardware emulators support highly detailed modeling (low level of abstraction) as they are similar to the final product but not truly the same. They are used effectively on large expensive projects, where the risk of failure is significant if some kind of prototyping is not done. Preparing a design for emulation should be considered a design project in itself.

Level 2: HDL model. In the full-system HDL model, the entire system on a chip, including the processor core, is simulated. Software compiled for the target processor is loaded into simulation memory and executed. The complexity of the model based on the number of modules will determine how fast the simulation can run. Typically, systems on a chip in full HDL will run between two and 10 processor instructions per second. This will not allow for much software testing. (I wouldn't want to try booting my RTOS in this mode.)

Software debug tools are not readily available for use at this level of abstraction. Since there is no "real

REAL TOOLS FOR THE REALLY HARD STUFF

We don't have to remind you how important it is to get your design into the market quickly and efficiently. But we would like you to know that our wide range of SoC verification tools excel at doing just that.

SoC VERIFICATION
TOOL SET

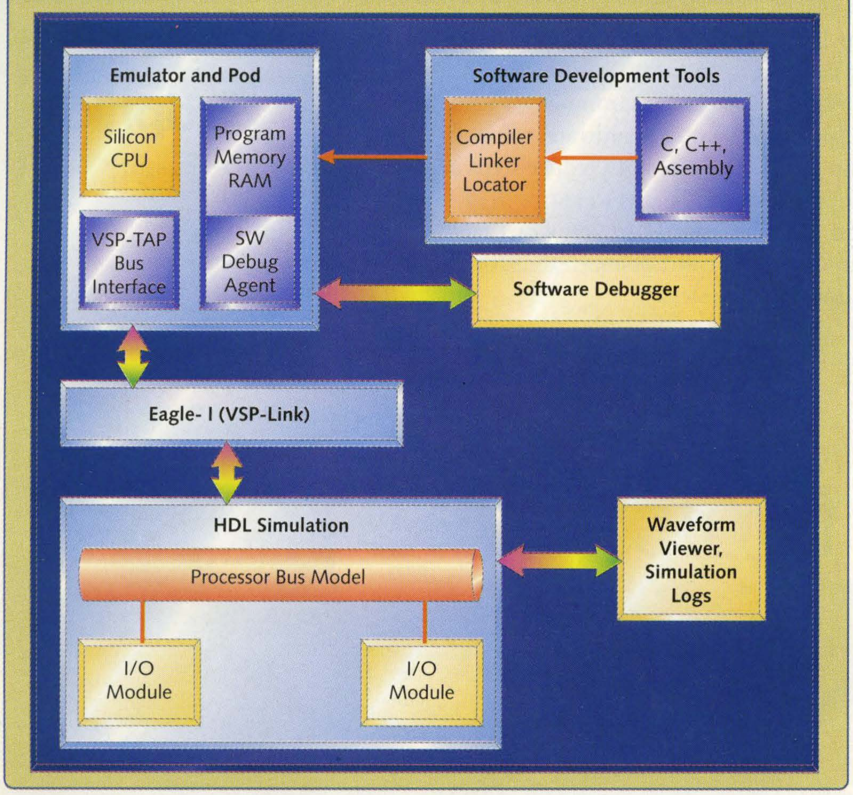
There's ModelSim®, for example, the leading tool for mixed language simulation. XRAY®, the only product to support debugging of multiple cores. Seamless® Co-Verification Environment, the industry standard by far. And a host of other hardware, software, IP and consulting solutions designed to help you increase productivity, reduce steps and slash verification time across every SoC development stage.

So be sure to start with our tools on the front end. Then you won't have to worry about the other one. For more information, call **1-800-547-3000**, or visit **www.mentor.com/soc** to get our new online SoC newsletter.

Mentor Graphics®

Our SoC verification tools

FIGURE 3 Abstraction Level 3: HDL logic with emulator



hardware," emulators can't be employed. Soft debuggers would have to be part of the running software. This is impractical due to the slow execution speeds. One possibility is to connect the host side of the debugger via a C-based interface to the simulation in order to get the required information back from the processor core for software debug. In most cases however, this is fundamentally a hardware debug environment.

Full system HDL models offer the most accurate platform available outside of the hardware emulator or prototype. The simulation can be highly detailed, running either RTL or gate level, depending on the verification requirements. Generally, this abstraction level is used by hardware and verification engineers to verify memory subsystems, critical timing, important hardware/software interactions, and the ability of the processor to boot and start executing code at reset. Figure 2 depicts the full system HDL abstraction level.

Level 3: Simulated logic in HDL with CPU emulator. In this scenario the ASIC logic remains in an HDL simulation. The processor model is removed and replaced with a specially designed interface that employs an actual in-circuit emulator of the real processor. The interface provides a trap mechanism that directs memory-mapped accesses of the ASIC logic to the simulation. The VSP-TAP and Core-TAP products from Applied Microsystems connect to an HDL simulation via the Synopsys Eagle co-verification tools.

Adding the real target CPU increases software execution to target speeds and provides the most accurate platform for software test and debug. Because of the fast software execution and virtually real CPU, it is possible to run large cross-compiled applications, including the RTOS. As such this platform can be used for full system cycle accurate verification. Total system throughput is gated by the frequency of callouts to the HDL simulation.

Figure 3 depicts the abstraction level 3 co-verification environment. Familiar EDA tools support hardware development and debug. Likewise, a commercially available software development environment is used for software design and debug. Hardware and software environments are linked together by a co-verification tool that handles synchronization and data transfers between the hardware and software development environments. In this case, the special "bus snooping" hardware is used to connect the in-circuit emulator to the co-verification tool.

Level 4: Instruction set simulator. In level 4 the CPU model is an instruction set simulator (ISS) for the target processor or DSP. The ISS is normally a C or C++ behavioral model of the processor. This model is at a higher abstraction level, as it does not detail the internal hardware structure (gates) like a full HDL model. Software can be developed using C/C++ or target processor assembly.

This configuration has two variations. In the first case the ISS is running in a separate process (in a Unix system) than the HDL simulation. The ISS provides a trapping mechanism that allows C function calls to be made when certain addresses are accessed by the executing software. Memory-mapped ASIC register accesses are trapped to function calls, which connect via foreign language interfaces to the HDL simulation passing read and write data to and from the software running in the ISS. Unix sockets are usually used as the physical interface between the ISS process and the simulation. This is a "non-cycle accurate" model as there is a loosely coupled interface between the running software and the logic simulation.

In the second case, an HDL wrapper surrounds the ISS, interfacing with the system bus. This provides a "cycle accurate" model where the

Any other way of developing software is MANUAL LABOR.

Why slave over writing, validating, debugging and documenting your code...when you can actually enjoy designing it?

We feel your pain. That's why we created Rhapsody®. It's the only visual UML compliant real-time application software development environment that simultaneously integrates and automates analysis, design, implementation and test.

Our exclusive Model-Code Associativity guarantees that your model and code are always in sync. Change your model, your code changes. Or–you won't believe this–change your code, your model changes. And you won't even break a sweat.

Frankly, Rhapsody is amazing. It automatically generates readable, deployable, production-quality C, C++ and Java®. Not just code frames, but ALL the code. You can say goodbye to the tedium of manually developing makefiles, state machines, communication infrastructures and the like. You can also move from one RTOS to another with the push of a button. Yes!

With Rhapsody's unique design-level debugging you can visualize, easily detect and instantly correct logic and programming errors. In fact, you can actually test your application as you build it. And, wonder of wonders, you can import and reuse your legacy code. How's that for flexible?

Rhapsody users are already reducing their development cycle by at least 30%. C'mon, what are you waiting for?

Rhapsody will change the way you work. Forever.

Visit us at www.ilogix.com to download a free copy of Rhapsody.

Rhapsody®

I-Logix®

FIGURE 4 Abstraction level 4: non-cycle accurate ISS with HDL

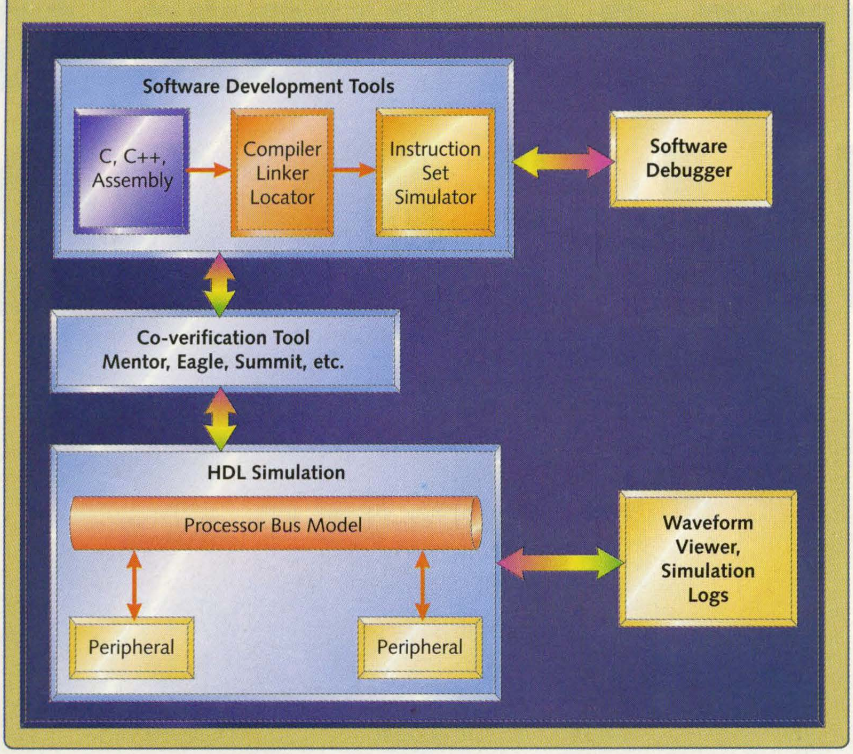
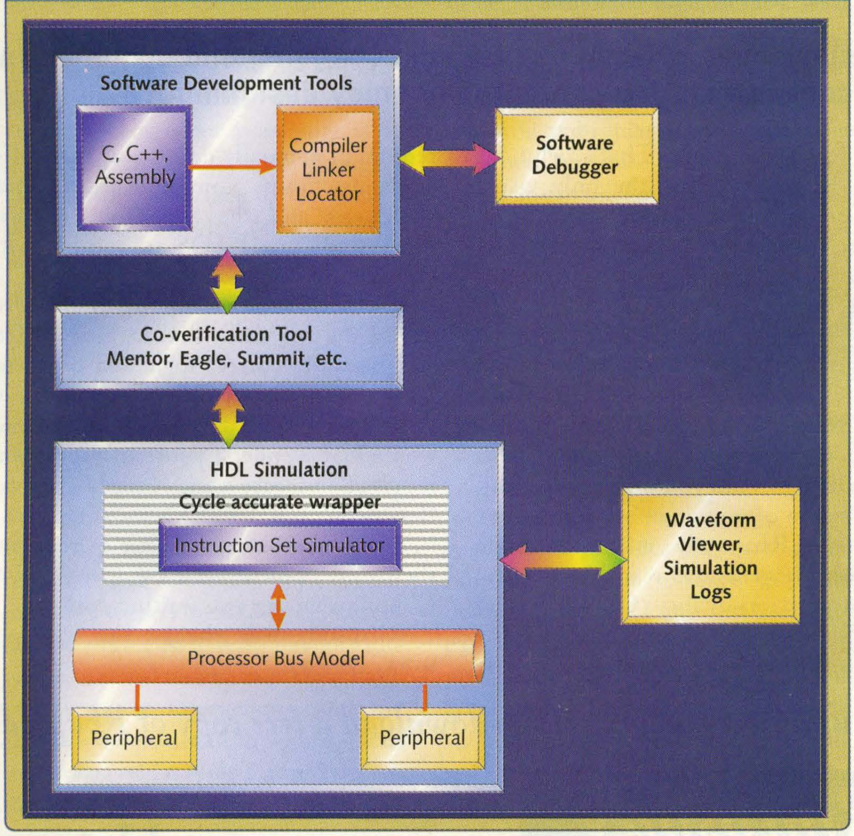


FIGURE 5 Abstraction level 4: cycle-accurate ISS with HDL

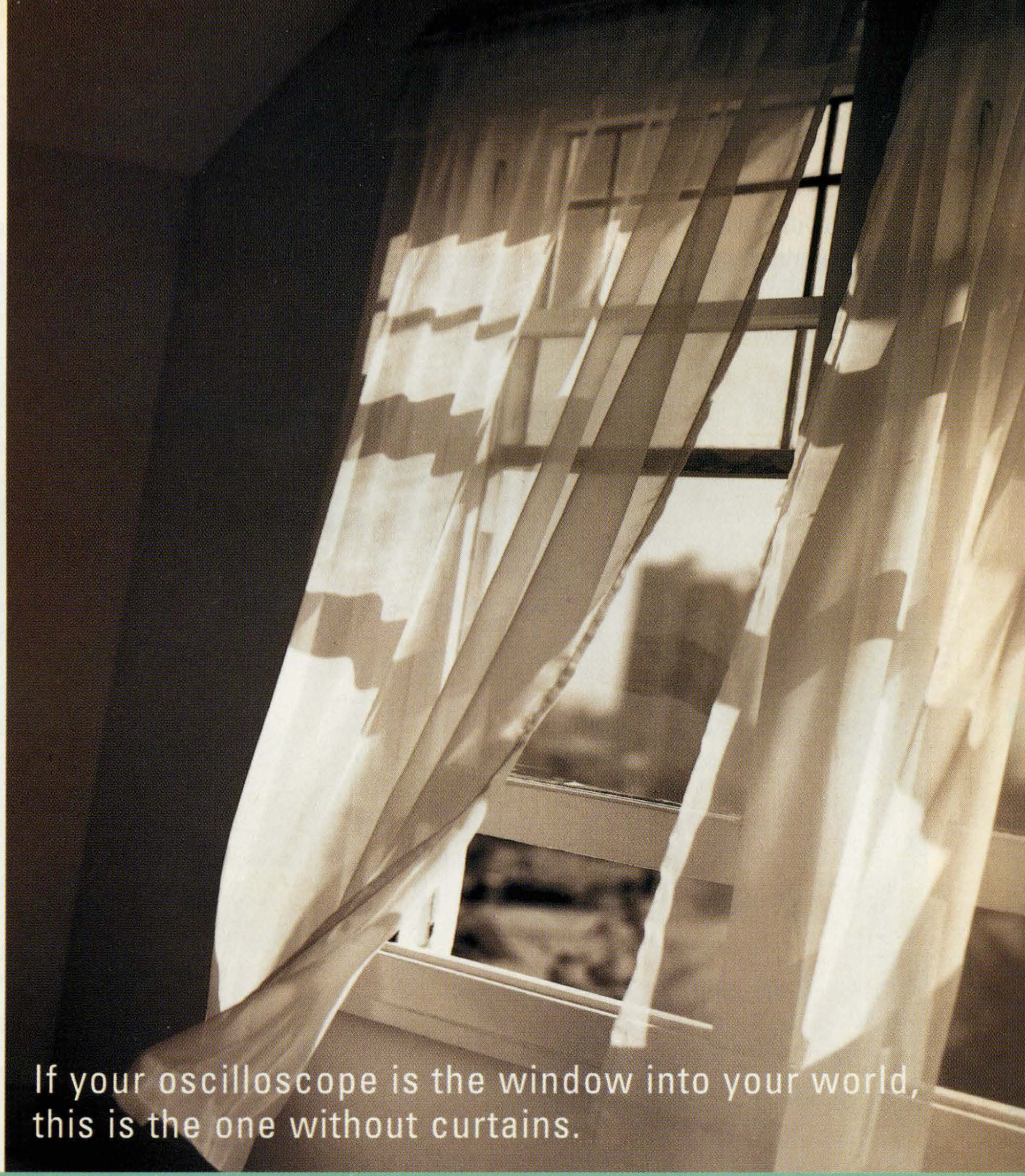


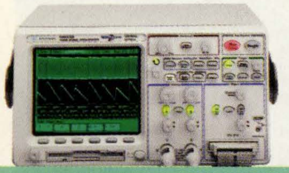
actual time (in system clocks) it takes for execution of each software op-code within the virtual CPU is maintained relative to the system bus cycle. The best ISS models support additional CPU internals such as instruction pipelining and cache operations.

This abstraction level offers a considerable software execution speed increase of a few orders of magnitude over a full system HDL implementation due to the elimination of the CPU hardware from the logic simulation. This is especially true of configuration one because of the de-coupling of the two environments. Performing program instruction fetches and data accesses out of ISS memory space (in configuration two) yields a sizable speed optimization even though the correct number of bus cycles are being simulated for each op-code execution. Eliminating actual memory accesses from the simulation greatly reduces the number of events in each simulation cycle. This level of abstraction allows a sizable amount of code to execute and, because of the possibility of cycle accuracy, supports software/hardware performance tests, interrupt processing, device driver development, RTOS installation, initialization, and test. This abstraction level is useful for module and system level test and verification.

Figures 4 and 5 depict two environments that will support HW/SW co-development at abstraction level 4. (Cycle accurate and non-cycle accurate.) Typically, a source-level software debugger runs within the ISS environment. Hardware and software development environments are linked together by a co-verification tool. Variants of these basic configurations are available from a number of EDA and software tools vendors.

Level 5: Host-native software. Level 5 increases software throughput dramatically by removing the CPU model entirely. A bus functional model


If your oscilloscope is the window into your world,
this is the one without curtains.



(BFM) on the HDL side is connected, via a co-verification tool, to software that is compiled from C or C++ for the host processor. The BFM is capable of executing atomic bus cycle such as read and writes to the ASIC registers using correct bus signal timing. It is also capable of returning interrupts to the host-based software for processing. Internal processor functionality such as pipelining and caching are not supported by the BFM.

FIGURE 6 Abstraction level 5: HDL logic simulation with host native software and BFM

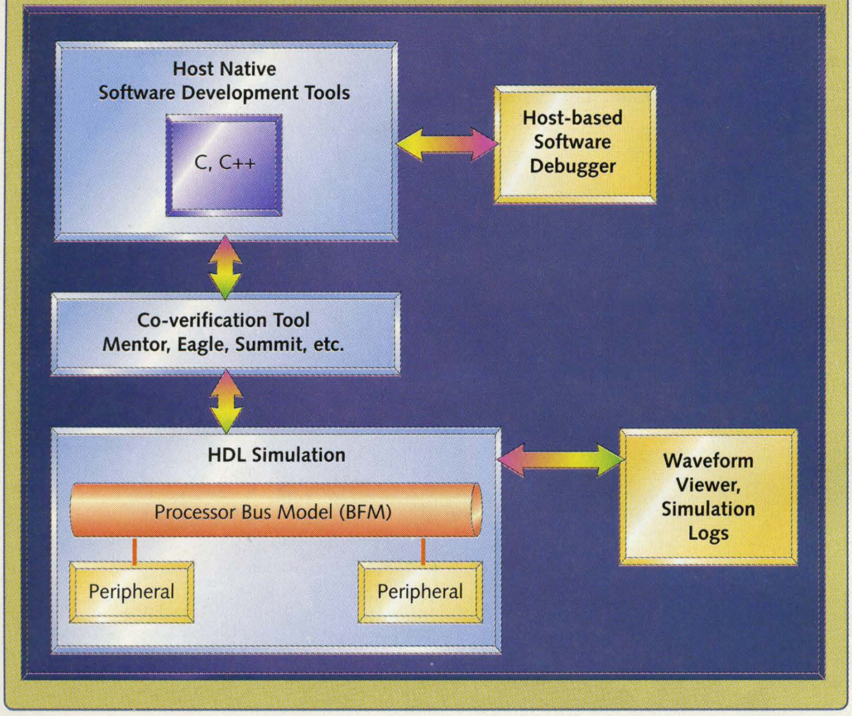
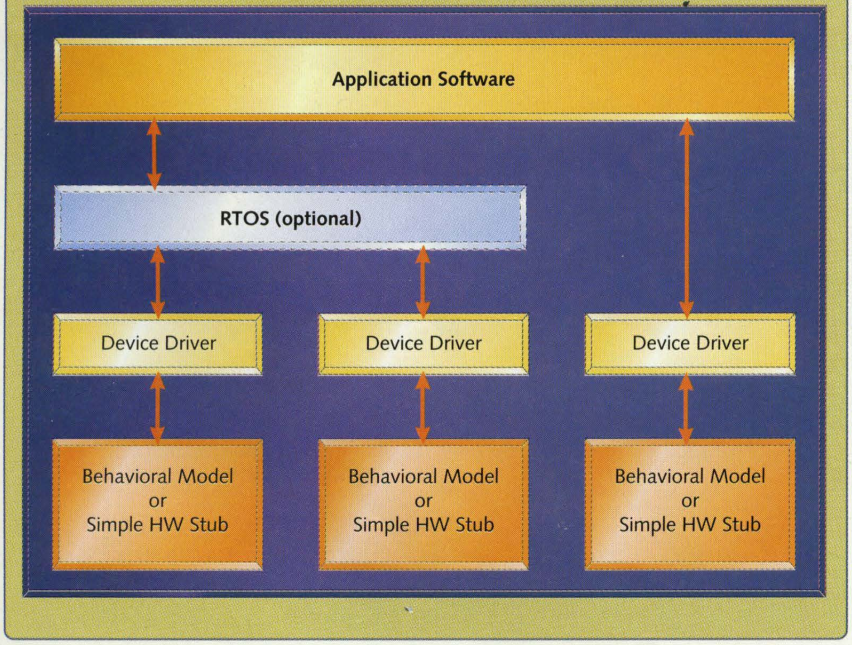


FIGURE 7 Abstraction level 6: Software-only development environment



The host platform provides an extremely fast execution engine for the software but lacks detail of the target CPU. Target assembly language is naturally not supported and there is no visibility into the processor register set or internal peripheral modules from the software side, as there is no processor model.

This level of abstraction is useful to hardware designers who wish to debug peripheral module functionality from the bus side by writing straightforward C programs to provide stimulus to the HDL logic. Software engineers can implement device drivers and diagnostics for peripheral modules. It's also useful for software engineers working at a high level, particularly on computationally intensive software, where throughput is a major concern. Additionally, a number of host-based RTOS simulators are available that can be used in this mode to bring whole RTOS-based application up with an interface to an HDL logic simulation. Figure 6 depicts a development environment using a co-verification tool that will support HW/SW co-development at abstraction level 5.

Level 6: Software only. The highest level of abstraction removes all hardware detail and relies on software behavioral models or simple stubs that pass data to and from a device driver. API's are used to detail the interface to the driver such that high level software can be written and tested at a structural level. No provision is made for timing of asynchronous events, so the real world interactions are abstracted away. As such, software throughput is excellent and, in some cases, faster than the eventual target, because of the horsepower of the workstation platform.

Got another date with
the old logic analyzer tonight?

COMPUTING | TELECOM | VIDEO

TLA 600 — starting at $7,000* You keep looking, but you can't find the problem. Time for a state-of-the-art, affordable logic analyzer that provides 500 ps timing simultaneous with up to 200 MHz state on every channel through the same probe? And a Windows interface? That's detail, accuracy, and ease that'll get you home in time for dinner. For a free Solutions Brochure, call 800-426-2200 x3039 or visit **www.tektronix.com/TLA600**

Tektronix®

*TLA 600 Series starts at $7,000 (manufacturer's suggested retail price). © 2000 Tektronix, Inc. All rights reserved.
Tektronix and the Tektronix logo are registered trademarks of Tektronix, Inc. All others are properties of their holders.

TABLE 1 Summary of system abstractions and their attributes

Abstraction Level	Simulation throughput	CPU	Attributes Cycle accurate	Models asynchronous events	Software	Target OS
6 Software only	Very fast	None	No	No	C, C++	Stub or simulation
5 HDL peripherals, host native software	Fast	BFM	No	Possible	C, C++	Stub or simulation
4 HDL peripherals, ISS	Medium	ISS	Maybe	Possible	C, C++, assembly	Real
3 HDL peripherals, emulator	Medium-fast	Real	Maybe	Possible	C, C++, assembly	Real
2 Full system HDL model	Very slow	HDL	Yes	Yes	C, C++, assembly	Not practical
1 Hardware emulation	Very fast	Real or bus model	Yes	Yes	C, C++	Real
0 Silicon prototype			Actual			

This level is useful for the early stages of software applications, algorithm development, and device driver development for real-time embedded systems. In the higher level application space, where interactions with hardware are minimal or event driven real-time processing involving concurrency are not an issue, this level may be used effectively by software engineers right up to HW/SW integration. This type of environment may be implemented based on an API interface specification supplied by the ASIC design team to the applications software design team early on in the project. Figure 7 depicts a software-only development environment.

Table 1 summarizes the attributes of the various levels of system abstractions discussed in the previous sections.

Using abstraction levels

The following is a brief summary of how hardware designers, software engineers, and verification engineers may use the different system abstractions presented in this article.

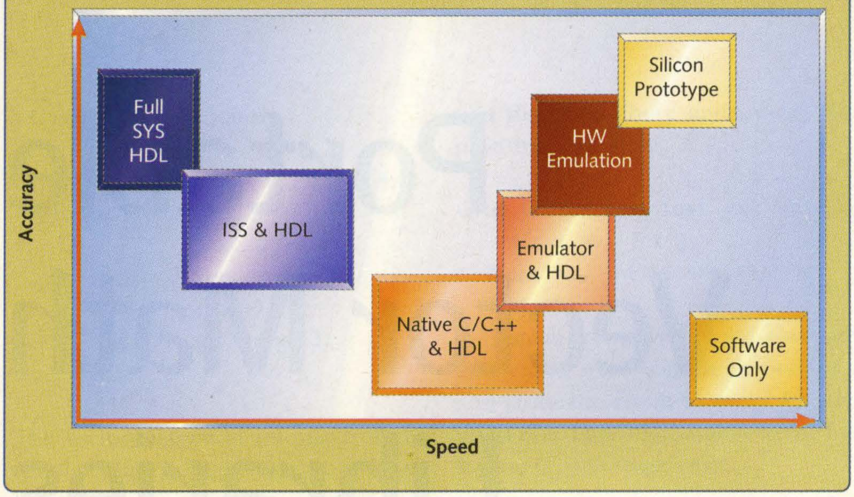
Hardware designers:

- Level 5: Debugging logic once removed from the CPU
- Level 4: Debugging logic touched by the CPU
- Level 3: Cycle accurate testing of software/hardware interactions
- Level 2: Debugging module interaction, verifying critical system timing and critical software/hardware interfaces
- Level 1: Fault grading, final regression tests, boot operating system
- Level 0: Final integration

Software engineers:

- Level 6: High level applications, APIs to device drivers
- Level 5: Applications, device drivers, diagnostics, RTOS simulation
- Level 4: Applications, RTOS bring up, device drivers, interrupts, when using target processor assembly code
- Level 3: All phases of software development. Most accurate software model, but may not be supported for your processor
- Level 2: Usually never, but may need to supply test code for hardware verification
- Level 1: Device drivers, applications, RTOS. Hardware emulation

FIGURE 8 System abstraction model accuracy vs. software execution speed



is expensive and may not be available for any given project

- Level 0: Final integration

Verification engineers:

- Level 5: Peripheral module verification
- Level 4: Interrupts, timers, internal memory verification
- Level 3: Easy test debugging while building regression test suite
- Level 2: Regression testing
- Level 1: Speedy regression

Execution speed vs. model accuracy

Figure 8 illustrates some of the tradeoffs in system accuracy vs. the amount of software throughput for the system abstractions discussed. Most system-on-a-chip design and verification work will be done using HDL simulations for the hardware. The type of processor model (HDL, emulator, ISS, or BFM), the level of detail of the logic (RTL or gates), and the frequency of accesses of software to the logic simulation ultimately determine the actual software throughput. Your mileage may vary.

System-on-a-chip designs are starting to become prevalent as advances in IC technology allow for ever higher levels of module integration. These devices will start to replace board level designs currently used for many high volume embedded systems. SOC designs combine traditional ASIC technology with embedded processor and DSP cores and memory. This requires close integration of software and hardware.

Co-verification tools allow for concurrent engineering of hardware and software. Hardware and software engineers use their standard development tools, which are linked via the co-verification tool environment. These tools allow the development team to work with various abstractions, or levels of detail, of the system being designed. Hardware and verification engineers can use low level software drivers to test the hardware. This re-use strategy can eliminate duplicate effort and bring in the schedule. **esp**

Bob Morasse is a technical manager at Intrinsix Corp., where he is responsible for research and application of co-verification technologies. Previously, Bob was a senior software engineer at Johnson & Johnson Clinical Diagnostics. Bob has over 14 years experience in embedded software design and hardware/software co-design and integration. His e-mail address is bmorasse@intrinsix.com.

WILLIAM WRIGHT AND JAMES METZGER

Portable Vector Math Libraries

Vector math libraries are used to perform calculations on arrays. Unfortunately, neither these libraries nor their APIs are very portable. This article compares and contrasts the different vector-processing capabilities that are available in hardware and suggests techniques for making vector libraries more portable.

A vector math library is software that application programmers use to perform mathematical calculations on an array of numbers. This array of numbers is also called a *vector*, hence the name "vector math library." Vector calculations are common in scientific applications such as sonar signal processing, medical imaging, and other areas that require the processing and analysis of sensor data in real time.

Because it represents the core of the computational software used by these applications, vector libraries are usually coded in assembly language for maximum speed and efficiency. Consequently, vector libraries are often bundled with a particular chip or bus board product. Some C coded libraries exist but these are less efficient and are usually used to prototype software on general-purpose computers.

The *vector-vector-add* operation is a basic operation typically found in a vector library package. If *x* and *y* represent two real-valued vectors of length *n*, the vector-vector-add operation adds the elements of *x* to the elements of *y* to produce a third real-valued vector, *z*. This would be represented in mathematical notation as follows:

$$z_i = x_i + y_i \qquad \textit{for } i = 0, n-1$$

This operation might look like this in

Given the similar requirements for vector math libraries for many processors, it's not surprising that a consortium is working to develop a platform-independent standard.

a C coded representation:

```
void vector_add(float *x, float *y,
float *z, int n)
{
   int i;
   for (i=0; i<n; i++)
        z[i] = x[i] + y[i];
}
```

The vector-vector-add routine from a vendor library might be prototyped like this:

```
void vector_add(float *x, float *y,
float *z, int n);
```

When using this `vector_add` operation in the calling routine, the programmer would allocate memory for the input arrays, *x* and *y*, and initialize them with the desired data. The programmer would also allocate memory for an output array, *z*, which the library routine will use to store the result. The variable *n* indicates the dimension of the arrays.

A typical vector math library will support basic arithmetic, trigonometric, and statistical operations. More comprehensive libraries can be found to support operations for signal and image processing applications, such as fast Fourier transforms and convolution. Also, libraries are not always limited to operations on floating-point vectors; some vendors also provide routines for matrices, complex data representations, and even integer arrays. Data type conversion routines are also quite common.

Another feature often supported by vector libraries is the concept of a vector *stride* that allows the programmer to control which elements of the vectors are manipulated. As an example, take the vector-vector-add function discussed above and modify the prototype as follows:

```
void vector_add(float *x, int i,
     float *y, int j,
   float *z, int k, int n);
```

This function would add every *i*th element of *x* to every *j*th element of *y* and store the result in every *k*th element of *z*. This is equivalent to the following C code fragment:

```
for (index=0; index<n; index++)
  z[index*k] = x[index*i] +
     y[index*j];
```

For a majority of uses, the vector stride values are set to one, representing a single element stride.

In general, the APIs of different vendors' libraries have similar anatomies, providing arguments for the address of the operands and the resultants, as well as a means for specifying the array dimensions. However, there is no agreed upon API standard to which all vector libraries adhere. Consequently, there are name-space conflicts, variations in stride support, different argument signatures, and specialized operations not found in all implementations. Unfortunately, this presents a problem to programmers that wish to support multiple platforms or ensure that the software they write will be portable to the next generation of general purpose DSP and its hand-coded assembly language utilities.

This article illustrates programming techniques designed to circumvent the problems associated with supporting multiple platforms and associated vector math libraries. These techniques were developed through porting and maintaining a real-time signal processing software system consisting of more than 300,000 lines of C code. The system consists of an executive component and a suite of signal processing modules that makes heavy use of vector computations. Since the signal-processing modules number in the hundreds, a brute force approach to supporting multiple architectures, operating systems, and vendors was both costly to port and impractical to test. And the prospect of tampering with already tested functionality summoned its own set of concerns. By using these programming techniques, we have been able to utilize vector math libraries from a variety of vendors for a variety of processors all while avoiding code changes to our processing module investment.

Standards

Given the similar requirements for vector math libraries for many processors, it's not surprising that a consortium is working to develop a platform-independent standard. But given the wide differences between the many processors used for signal processing, it's also not surprising that developing such a standard is a difficult job.

The Vector, Signal, and Image Processing Library (VSIPL) Forum is a consortium developed out of a research project by the U.S. Defense Advanced Research Project Agency (DARPA). The goal of the consortium is to develop an API for signal processing that is portable across platforms, allowing application developers to write signal processing software that can be used in many environments. The consortium members include academics and vendors who make signal processing hardware and software used in military applications. The consortium began work on the VSIPL API in 1996 and maintains a Web site at *www.vsip.org.* They

BEN FISHMAN

LISTING 1 Vector-scalar multiply routine

```
/* SKY Standard Math Library */
vsmul(float *invec, int j, float scalar,
float *outvec, int k, int n);

/* SKYvec library does not support strides and uses a global */
/* variable for the vector length */
_skyvec = n;
v$_rsvt0(float scalar, float *invec, float *outvec);

/* Wideband Computers Inc (SHARC) */
vsmul(float *invec, int j, float scalar,
float *outvec, int k, int n);

/* Alacron i860 */
vsmul(float *invec, int j, float scalar,
float *outvec, int k, int n);

/* Alacron SHARC(passes scalar by reference)  */
vsmul(float *invec, int j, float *scalar,
float *outvec, int k, int n);

/* Intel Native Signal Processing (NSP)for Pentium   */
/* Operation done in place. Does not support stride  */
nspsbMpy1(float scalar, float *vec, int n);

/* Mercury Computers(passes scalar by reference) */
vsmul(float *invec, int j, float *scalar,
float *outvec, int k, int n);
```

released the version 1.0 specification in the fall of 1999. However, no implementations of the library are widely available.

Because most signal processing applications use only a small subset of the universe of vector operations, engineers who need portable code can define their own vector library API taking their application requirements into account. This gives the application developer the flexibility to choose what sacrifices are appropriate in terms of execution speed and memory usage to achieve portability.

Preprocessor macros

One technique that can be used to achieve greater portability when writing code to support vector math libraries is using preprocessor macros to define an API that encompasses the variants of the libraries. As an example, take the case of a vector math operation that multiplies each element of a floating-point array with a single floating-point scalar. This is commonly referred to as a *vector-scalar multiply*. This operation might look as follows in a C code fragment that supports vector striding:

```
void vsmul(float *invec, int j,
     float scalar, float *outvec,
     int k, int n);
{
  for (i=0; i<n; i++)
     outvec[i*k] = scalar*invec[i*j];
}
```

Listing 1 shows the API of the same operation in various libraries used at some point in our development.

Depending on the implementation, the scalar is passed either by value or by reference. Some APIs do not support vector striding and others will only do the computation in place. For signal processing software that typically requires a sequence of vector math library calls, one can imagine the unreadable complexity of a software module written to support multiple targets in a brute force approach using preprocessor directives, one for each of the vector math libraries supported by the code. Listing 2 shows an example.

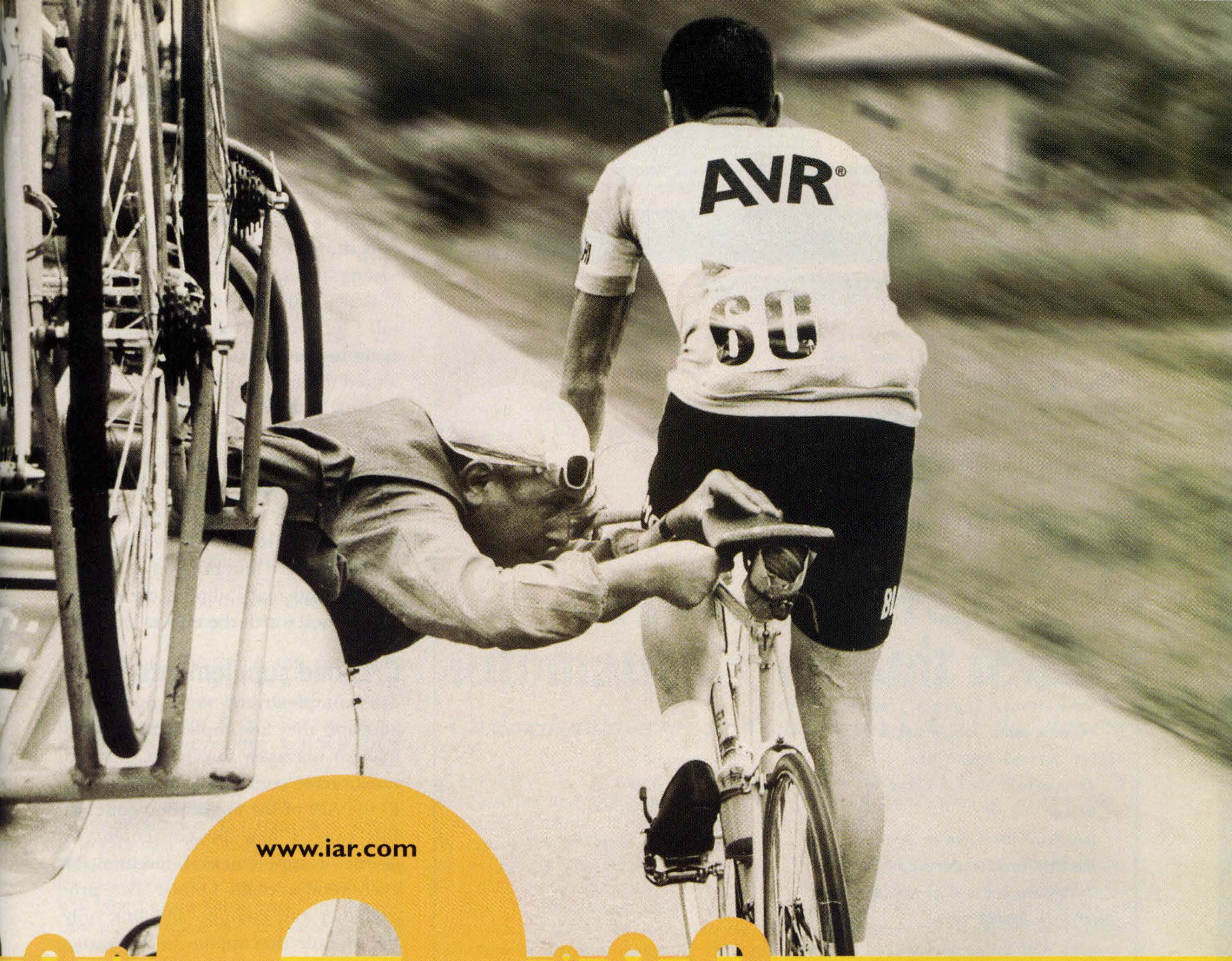
As Listing 2 shows, a block of code has to be added to the existing module for each new target supported. Furthermore, any new processing module developed would require a number of `#ifdef` code sections in order to support multiple architectures. Add to this the `#include` statements needed to properly prototype the functions and the code becomes difficult to maintain. There is, however, a better way.

The software developer should initially adopt or define an API that encompasses the essential functionality of the vector math library. In our real-time signal processing system, many of the signal processing modules had been first written and tested to a particular vendor library which had prototypes like this:

```
vsmul(float *invec, int j, float
    scalar, float *outvec, int k,
    int n);
```

We defined a new header, `myvec.h`, designed to isolate all the porting mechanisms from the standard C module that uses the math functions, and provide the translation between the programmer's API and that defined by the vendor. Used in the implementation, the module from Listing 2 looks a lot cleaner:

```
#include <myvec.h>
```


AVR®
60
www.iar.com

LISTING 2 Brute force approach to supporting mulitple vector math libraries

```
#if defined(VENDOR_A)
#include <aveclib.h>
#elif defined(VENDOR_B)
#include <bveclib.h>
#endif /* VENDOR */

/* etc */

my_processing_module()
{
/* code */

#if defined(VENDOR_A)
  /* other vector calls */
  scalar = 2.0f;
  vsmul(invec, 1, scalar, outvec, 1, n);
  /* even more vector calls */
#elif defined(VENDOR_B)
  /* previous vector calls */
  scalar = 2.0f;
  bvecCopy(invec,outvec,n);
  bvecMpy1(scalar,outvec,n);
  /* subsequent vector calls */
#endif /* VENDOR */

  /* other architectures, etc. */
}
```

```
my_processing_module()
{
    float scalar = 2.0f;
    vsmul(invec, j, scalar,
        outvec, k, n);
    /* etc */
}
```

Conversely, inside the `myvec.h` header file we have all the ugliness with the preprocessor directives, albeit well organized into segments:

```
#if defined(VENDOR_A)
#include <aveclib.h>
#define vsmul vsmul
#elif defined(VENDOR_B)
#include <bveclib.h>
#define vsmul(a,i,b,c,k,n) \
  {bvecCopy(a,c,n); bvecMpy1(b,c,n);}
#endif /* VENDOR */
```

At compile time, the C preprocessor will substitute the `vsmul` vector call for the `vsmul` symbol in the case of the VENDOR_A target. In the case of the VENDOR_B target, the `vsmul` symbol is expanded to a set of optimized function calls arranged to duplicate the operation of the adopted API.

This approach is beneficial in a number of ways. First, the module that uses the portability API does not have to be modified when porting to another platform. For modules that have undergone acceptance testing and have a reliable track record, this is an important benefit since it minimizes the potential introduction of bugs into the code. Validation of the substitution macro is the only thing that needs to undergo regression testing (see the Regression Testing section near the end of this article).

Second, adding functionality to the module is a matter of adding function calls drawn from the defined API. By writing the target-independent code in this manner, all other architectures will be supported automatically, assuming the function calls are represented for each target.

Finally, supporting future targets is simply a matter of adding a new `#ifdef` block to the `myvec.h` header file corresponding to the new target and the vector math library it will use. Although this is still a bit of work, it is straightforward and the overall ease of porting and readability of the source code is well worth the effort.

C-coded supplements

The complexity of some operations and how they are implemented often make it necessary to rely on C coded functions to conform a vendor library to the API of the application code. In this case it may be convenient to define a supplemental module for each vendor math library. The programmer can compile and link only the module that applies to the particular target being addressed.

For example, some of the more obscure vector calls, such as `vsmod`, used in our real-time signal processing application did not have a one-to-one mapping with any vector routines in a particular vendor library. To duplicate the *vector-scalar-modulo* functionality using the vendor's vector calls required multiple operations and storage for intermediate results (Listing 3).

This routine and other special case routines were placed in a file (for example, `vendorplus.c`) and prototyped in a header file (`vendorplus.h`). In order to minimize impact on the application level code, references to these functions can be made in the `myvec.h` header. This way the `vendorplus.h` header does not need to be included in the application level code, but only in `myvec.h`, regardless of the target of interest.

As much as possible, developers should cast the C coded routines into

Paid advertisement

Automatic Unit Testing

by Michael Aivazis

If you've read this column before, you know that we are advocates of unit testing and other forms of error prevention. However, we also acknowledge that there is a major obstacle to performing unit testing: the fact that we are human. As humans, we don't like to take on tasks that are time consuming or that increase our workload. Unit testing does seem tedious at first, and those who perform the process manually are likely to give up entirely because of the time involved.

On the other hand, the fact that we are human is one of the best reasons to automate unit testing. When we perform manual unit testing, we run the risk of making mistakes simply because we are human. After all, manual unit testing is a complicated process because we must spend a good deal of time thinking about what inputs we need to test a class.

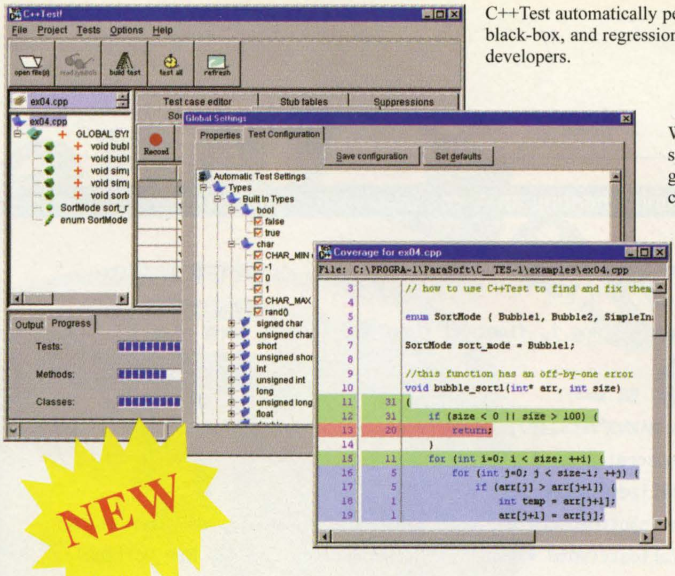
Aside from saving time and preventing human errors, the best reason to automate unit testing is to facilitate regression testing. Regression testing is a method of preventing yourself from introducing new errors to code as you make repairs. Developers often think regression testing can only apply to an entire application, but regression testing is in fact very effective at the unit level. When they find errors in a class, it is easy for developers to create a test case that will guard against that error, put the test case in their regression test suite, and run the suite.

Every time you modify your code, you should run your code against your regression test suite to make sure that your code has not "regressed" into previous errors. When you find an error, you can write a test case against it and add it to your suite. As you develop your application, your regression testing suite will grow.

There is no limit on regression testing, and the more you perform it, the greater the benefits for your code. You can even use a script to automatically pull up a class, compile it, and run it against your test suite. You'll get a clear idea of what is going on with the code, and you don't even need to be in the office when the testing occurs.

These are just a few of the benefits of automatic unit testing. Fortunately, it is possible to automate unit testing on any development platform. You can use scripts, or select from a variety of automatic tools. No matter what method you use to automate testing, you will save time as you improve the quality of your applications.

Michael Aivazis, Ph.D., is Director of Technology at ParaSoft. You can reach him at mga@parasoft.com.



ex04.cpp
- GLOBAL SY
  - + void bubl
  - + void bubl
  - + void simj
  - + void simj
  - + void sort
  - SortMode sort_r
  - enum SortMode

Automatic Test Settings
- Types
  - Built In Types
    - bool
      - ☑ false
      - ☑ true
    - char
      - ☑ CHAR_MIN
      - ☑ -1
      - ☑ 0
      - ☑ 1
      - ☑ CHAR_MAX
      - ☑ rand()
    - signed char
    - unsigned char
    - short
    - unsigned shor
    - int
    - unsigned int
    - long
    - unsigned long
    - float

```
File: C:\PROGRA~1\ParaSoft\C__TES~1\examples\ex04.cpp
 3        // how to use C++Test to find and fix them
 4
 5        enum SortMode { Bubble1, Bubble2, SimpleIn:
 6
 7        SortMode sort_mode = Bubble1;
 8
 9        //this function has an off-by-one error
10        void bubble_sort1(int* arr, int size)
11    31  {
12    31      if (size < 0 || size > 100) {
13    20          return;
14            }
15    11      for (int i=0; i < size; ++i) {
16     5          for (int j=0; j < size-i; ++j) {
17     5              if (arr[j] > arr[j+1]) {
18     1                  int temp = arr[j+1];
19     1                  arr[j+1] = arr[j];
```

C++Test automatically performs white-box, black-box, and regression testing for C and C++ developers.

With C++Test, you can specify test parameters globally or for the current project.

C++Test's coverage analysis feature graphically displays the lines of code that are being tested.

It's finally here... automatic C/C++ unit testing!

Customizable error prevention for C/C++ developers

C++Test™ is a new tool from ParaSoft that automates C and C++ unit testing, making it easier to find and fix errors before they mushroom into more serious problems. Testing with C++Test is automatic, but it leaves you in control. For the quickest testing, you can click once to have C++Test open, build, and perform white-box testing on your files one after another. For more complete testing, you can modify test parameters and let C++Test perform automatic white-box, black-box, and regression testing.

White-box testing tests the construction of your code. C++Test performs white-box testing by automatically generating and executing test cases to see how files, classes, or methods behave when they are passed unexpected inputs. You can control what types of test cases C++Test generates by specifying what kinds of arguments to use, how many test cases to run, and how deeply to search your embedded classes. C++Test will also build stub functions to test methods that call other functions.

Black-box testing verifies that each class behaves according to specifications. C++Test performs black-box testing by automatically generating and executing test cases and displaying the outcomes, which you can verify with one click of the mouse. You can also enter your own black-box test cases to be executed by C++Test.

Regression testing verifies that your modifications corrected problems and did not introduce new problems into your code. C++Test performs regression testing by executing the test cases that were run the previous times the class was tested and checking to see if any outcomes have changed. If outcomes have changed or if C++Test finds exceptions, it will report errors. You can replay the test cases each time you modify your code.

Of course, unit testing is just one stage of the error prevention process for C and C++ developers. C++Test works with ParaSoft's CodeWizard® to enforce coding standards automatically each time you build a file. C++Test also works with ParaSoft's Insure++® to give you automatic runtime error detection each time you test a class or method. Ask about ParaSoft's specialized Developers Kits for your development team.

If you would like the satisfying experience of testing automatically at the unit level, please take a moment to download a fully-functional demo copy of C++Test today at www.parasoft.com/best.htm, or call us at (888)305-0041 for more information.

ParaSoft®

www.parasoft.com/best.htm

LISTING 3 Implementing an uncommon vector function

```
static float nspscratch[MAXNSPVECSIZE];

void vsmod(float *a,long i, float s, float *c, long k, long n)
{
      assert(i==1 && k==1)
      assert(n<=MAXNSPVECSIZE);
      /* use nspscratch as a often = c */
      nspsbNormalize(a,nspscratch,n,0.0,s);
      nspsbFloatToInt(nspscratch,(void *)nspscratch, n, 32, NSP_TruncZero);
      nspsbIntToFloat((void *)nspscratch, nspscratch, n, 32, NSP_Noflags);
      nspsbMpy1(s, nspscratch, n);
      nspsbSub3(a, nspscratch, c, n);
}
```

the vendor vector library calls since these are assembly coded, optimized routines.

Name space conflicts

Occasionally, two vector library vendors will use the same name for functions that perform different operations. This makes it impossible to define a cross-platform API using that name without causing compile or link errors or incorrect operation on some targets. The solution is to define a new name for the operation different than all vendor libraries. For example, vendor A might define a function called vadd() that adds two vectors and vendor B might define vadd() that adds a scalar to a vector and vvadd() that adds two vectors. In this case you can avoid the problem by defining a new function that clears up the ambiguity. Then define the new function in terms of the vendor functions. Defining a prefix or suffix to be used on all of the new functions can be useful for avoiding name space conflicts. This example uses the suffix "_sys" for each function:

```
#if defined(VENDOR_A)
#define vectorVectorAdd_sys(/* \
  arguments */) vadd(/* \
  arguments */)
#elif defined(VENDOR_B)
#define vectorVectorAdd_sys(/* \
      arguments */) vvadd(/* \
      arguments */)
#endif /* VENDOR */
```

Preprocessor issues

The C preprocessor is a powerful tool for generating targeted code from a platform-independent baseline, but there are some common pitfalls that can cause problems. One is the use of the auto-increment or auto-decrement operator. If a macro function expands using an argument more than once, the operator is evaluated multiple times causing unexpected results. For example the macro function:

```
#define vcopy(a, b, c) copy(a, \
  a, b, c)
```

when called like this:

```
vcopy(src++, dest, size);
```

causes the first argument "src" to be incremented twice since the preprocessor expands the expression to this:

```
copy(src++, src++, dest, size);
```

To avoid this problem, don't use auto-increment in an argument list but rather place the increment operation after (or before, for a preincrement) the vector operation:

```
vcopy(src, dest, size); src++;
```

Unfortunately, it's difficult to enforce this discipline on all programmers.

Another potential problem occurs when macro functions expand to multiple statements. Their behavior within conditionals needs special consideration. For example, the macro function:

```
#define vcopy(a, b, c) \
  vsize=(c); vmove(a, b)
```

can have unexpected behavior when used after an unbracketed conditional:

```
if (size > MINSIZE)
  vcopy(src, dest, size);
```

The code above expands to this:

```
if (size > MINSIZE)
  vsize = (size);
vmove(src, dest);
```

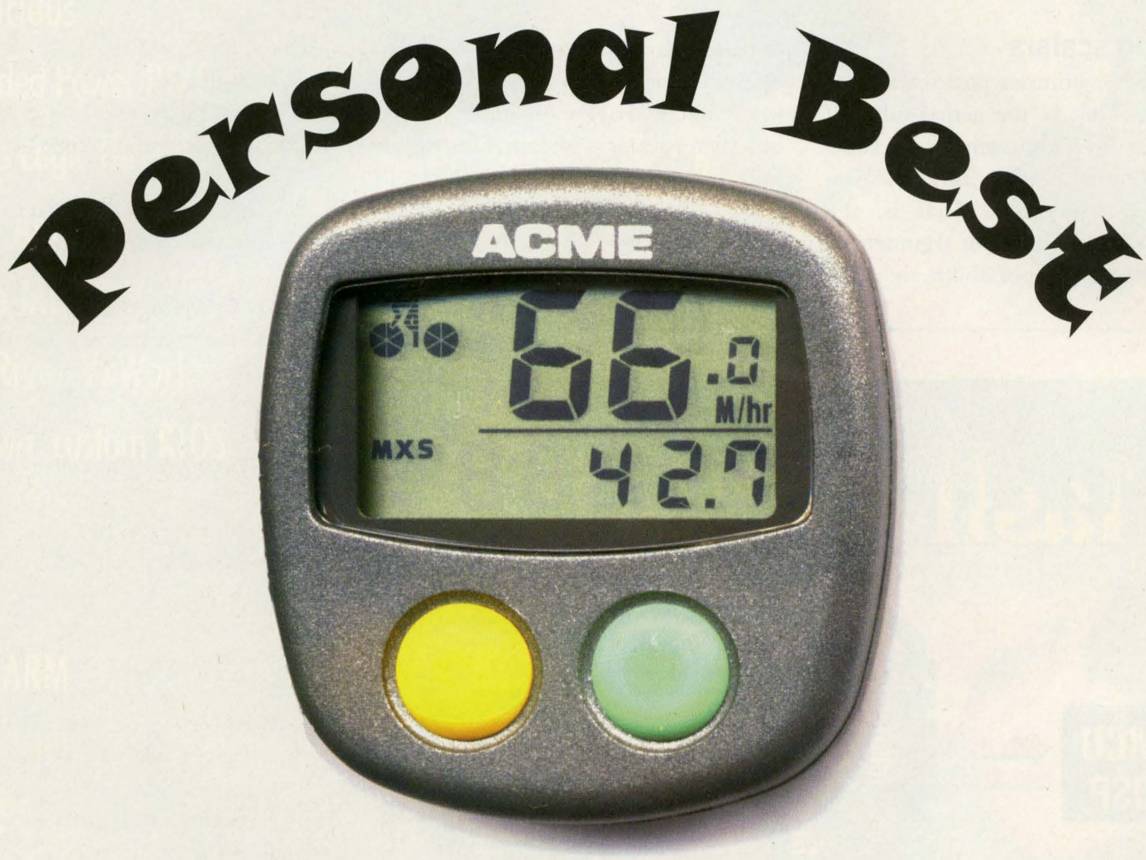
Clearly this is incorrect. To avoid this problem, enclose multi-statement expansions in braces like this:

```
#define vcopy(a, b, c) \
{vsize=(c);vmove(a, b);}
```

Also, the misuse of semicolons in the macro statement can sometimes cause warnings during compilation. Notice that in the following definition, the end semicolon is omitted from the last statement:

```
#define vsmul_sys(a,i,b,c,k,n) \
  vecscalmul(b,a,i,c,k,n)
```

After the preprocessor substitution there will not be two semicolons in the preprocessor output.



There's a feeling about achieving a goal, or simply reaching a new level of performance that's hard to put into words. It's not about winning or beating the next guy necessarily, just about knowing that you've done more than you've ever done before.

HI-TECH C users get that feeling often - the feeling of being "in the groove", the realization that not only did you meet your deadline, but you did it easily, without stress.

If you program embedded microcontrollers, and you want to raise your performance level, get HI-TECH C. World-beating performance from a tool that gets the job done without getting in your way, and backed by support our competitors can't believe - that's what *you* need to achieve your personal best. Call us today, or visit our web site for demos and information.

www.htsoft.com

HI-TECH Software
7830 Ellis Rd.
Melbourne FL 32904
Ph 800 735 5715
Fax 407 722 2902

HI-TECH
SOFTWARE

Passing scalars

Some vector libraries pass scalar data by value. That is, the actual value of the scalar is an argument to the functions. Other vector libraries pass scalars by reference. That is, the address of the scalar is an argument to the functions. For portability, we chose to pass scalars by value and map them to references with a macro function when necessary. For example, these two vector-scalar add functions differ only in how the scalar is passed:

```
/* VENDOR_A.h */
void vsadd(float scalar, float
  *src, float *dest, long size);
/* VENDOR_B.h */
void vsadd(float *scalar, float
  *src, float *dest, long size);
```

We could define the macro function for vendor B this way:

```
#if defined (VENDOR_B)
#define vsadd_sys(s, a, c, n) \
  vsadd(&s, a, c, n)
#endif /* VENDOR_B */
```

This works fine for this call:

```
float scalar = 100.0f;
vsadd_sys(scalar, src, dest, 10);
  /* "scalar" is a float */
```

which expands to

```
vsadd(&scalar, src, dest, 10);
  /* Pass the address scalar */
```

The problem comes when a constant is used for the scalar:

```
vsadd_sys(100.0f, src, dest, 10);
```

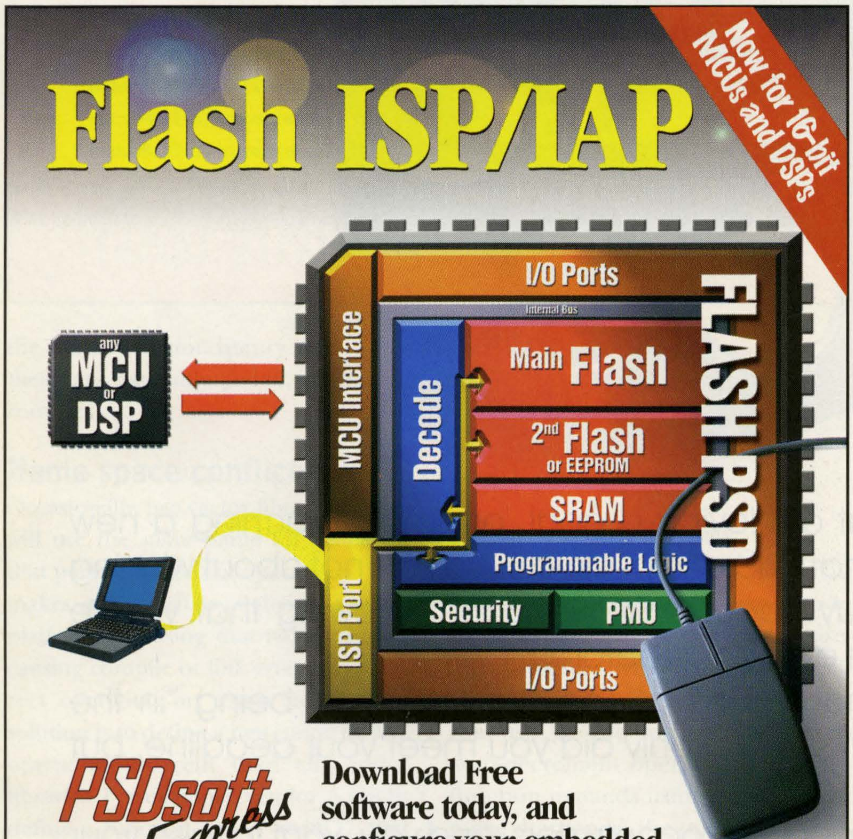
which expands to:

```
vsadd(&100.0f, src, dest, 10);
  /* Syntax error */
```

The solution is to use a temporary variable for pass-by-reference macro functions:

```
#define vsadd_sys(s, a, c, n) \
{float tmp = s; vsadd(&tmp, a, \
  c, n);}
```

If a particular C compiler doesn't support declaring a variable within a block, this technique can still be used. Declare a global temporary variable and assign the scalar to it. Then pass the address of the temporary variable to the vendor function. Naturally, this puts limits on the use of vector functions in multithreaded code or interrupt service routines. If more than one thread of control uses it, the shared global variable could be cor-



rupted if reused before the function finishes.

Quirks

Sometimes the constraints of the hardware or the implementation of the vector library require the source and destination buffers to be non-overlapped. So, calls like:

```
vadd_sys(v1, v1, size);
```

will yield incorrect results. The solution is to either avoid in-place operations altogether or define a wrapper function that copies the source vector to a temporary buffer so the out-of-place operator can run correctly.

Strides are another area where differences in vector library capabilities appear. Some libraries have a rich support for strides (some even support negative strides) while others assume a stride of one for all operations. This implies that targeting a library without strides should mean minimizing the use of strides in the application code. On the other hand, operations that must use strides will not be able to use vector libraries that do not support striding and therefore will either be slower (because they are written in C) or hard to write (because they must be coded in assembly language). If the application making the vector function calls always uses a stride of one, a simple solution is to create macros that don't reference the stride arguments. But what if somebody decides to use a stride later? The results of the macro function will be incorrect. Catch these problems by using the assert macro within the definition of the macro function:

```
#define vcopy_sys(a, as, b, bs, n) \
  {assert((as==1)&&(bs == 1)); \
  vcopy(a, b, n);}
```

This way the function will report the error rather than just making the wrong calculation.

Libraries that support strides often differ in how they interpret strides on complex vectors. A complex value is usually implemented as a struct containing the real and imaginary values like this:

```
typedef struct complex {
  float real;
  float imag;
} Complex;
```

So a vector of complex type is a sequence of float pairs. Some libraries define complex strides in words and some define strides in units of the `Complex` struct. That is, assuming a float is four bytes, some libraries interpret a stride of one as a four-byte stride and some interpret it as an 8-byte stride. Striding by half the size of the `Complex` struct is rarely the desired behavior. We recommend defining a portability API using a stride of one to represent a single (8-byte) complex stride. For libraries that use four-byte strides a macro like this will do the conversion:

```
#define cvadd_sys(a, as, b, bs, \
  c, cs, n) \
  cvadd(a, (as)*2, b, (bs)*2, c, \
    (cs)*2, n)
```

Complex data types

Vector libraries differ widely in how they treat the complex data type. Some libraries define their own struct for complex data and their functions expect vectors of that type as arguments. Other libraries just treat a float vector differently by assuming that it is made up of alternating real and imaginary values. This variety can make writing portable application code difficult. Since most libraries fundamentally expect a vector of alternating real and imaginary values, the problem is really one of casting. That is, the complex vector needs to be cast to the type expected by the vector operation.

We addressed this problem by defining our own complex struct and defining all of our complex vectors in terms of our struct. Then we defined a macro to represent a vendor-independent cast to the type that the vendors' functions expect. This macro is used before each complex argument in the application code.

```
cvadd_sys((CMPLX *)a, 1, (CMPLX
  *)b, 1, (CMPLX *)c, 1, n);
```

This macro is defined differently depending on the library's expected type:

```
#if defined(VENDOR_A)
#define CMPLX SCplx

#elif defined(VENDOR_B)
#define CMPLX float
#endif /* VENDOR */
```

This way the vectors are of the correct type for each library and we take advantage of complex type checking if it is available.

Obscure vector calls

Vector libraries differ greatly in the number of operations they provide. Some provide only the most basic arithmetical operations while others have functions for intricate domain-specific calculations. Powerful, higher level operations can be efficient, but may not be available on every platform. Some libraries support vector operations on many types including integer, boolean, and floating-point types, but many do not. For targets that don't support needed operations, they must be implemented within the portability library. They can often be implemented efficiently using the simple vector operations that are supported. But some frequently used operations may need to be coded in assembly language to get the needed performance.

Fourier transform

The discrete Fourier transform (DFT) is a common mathematical operation used in engineering applications. The following discussion on DFT vector library routines addresses the one-dimensional transform, but similar principles apply to multi-dimensional

The Perfect Message For Embedded Systems Developers.

Now that Metrowerks is part of Motorola, embedded systems developers can take advantage of the most timely, tightly integrated, total solution for embedded systems development.

Metrowerks' CodeWarrior, with its award-winning IDE and open architecture, provides superior software development tools for the most powerful and versatile Motorola processor architectures, including DSP, PowerPC,™ M-CORE™ and 68K/ColdFire.®

Free 30-Day Evaluation of CodeWarrior

Call today or order online

1-800-377-5416

www.metrowerks.com/go/esp

sales@metrowerks.com

CodeWarrior for PowerPC Embedded Systems

The newest release features:

- Support for the entire family of PowerPC processors, including 5xx, 6xx, 7xx, 8xx, 82xx, and AltiVec™ (MPC 7400)
- Command-line tools and makefile importer to enable developers to use legacy code
- Public API enables kernel awareness for commercial or proprietary RTOSes

metrowerks

A Motorola Company

CodeWarrior.

LISTING 4 Different FFT initialization routines

```
/* Wideband Computers Inc. for 2106x */
/* requires that the real coefficients reside */
/* in different memory spaces */
void fftwts(float pm *wr, float dm *wi, int N);

/* Alacron 2106x */
void init_fft_sh(int N);

/* Intel Native Signal Processing */
void nspsRealFftNip(float *in, SCplx *out, int order, int flag);
```

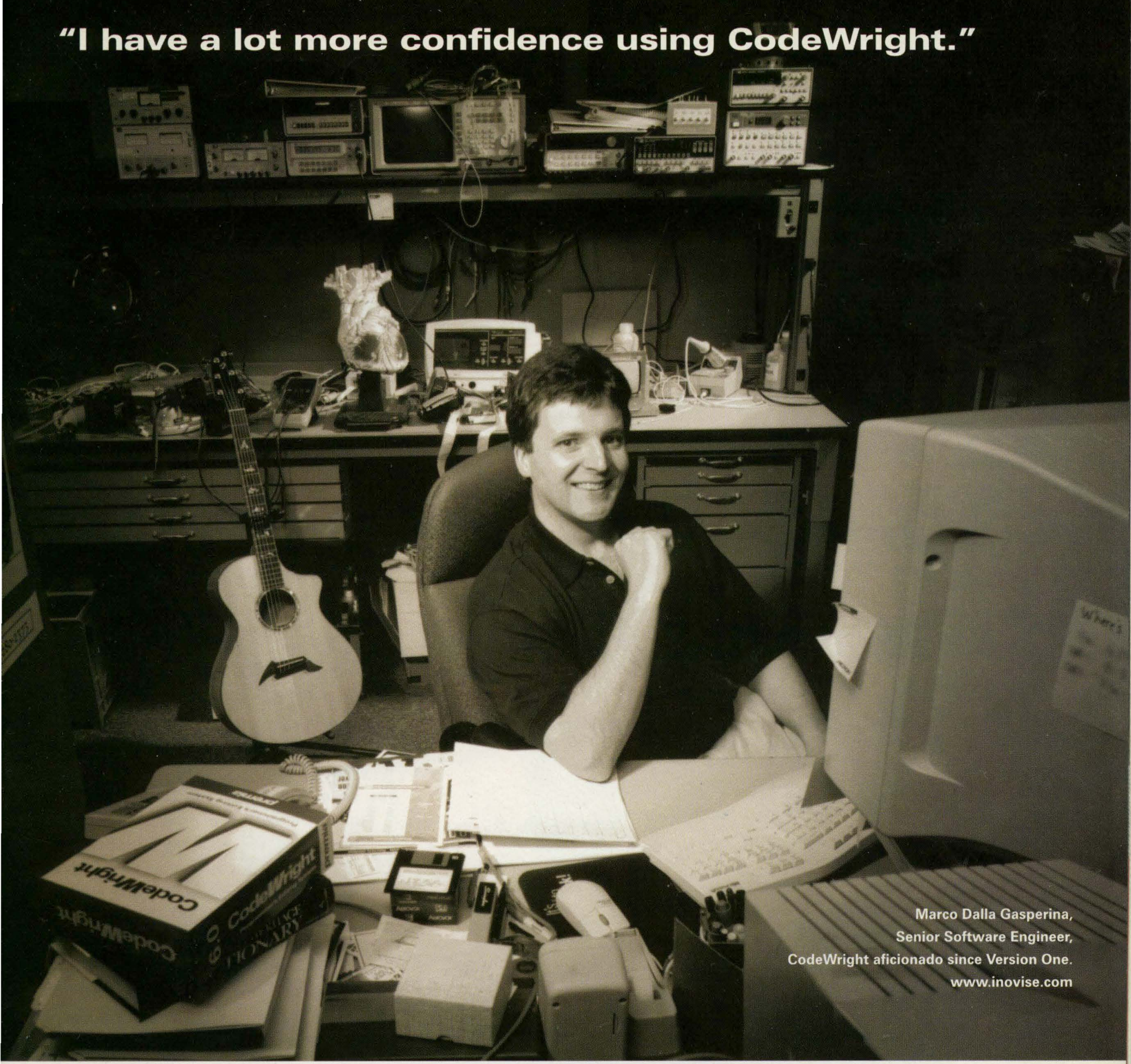
transforms. The basic operation takes a vector $x[n]$ of length N and computes another vector $X[k]$ of the same length using the following formula:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi nk}{N}}$$

for $k = 0,\ldots, N\text{-}1$. A reciprocal operation, called the inverse DFT, has a similar structure:

$$x[n] = \frac{1}{N} \sum_{n=0}^{N-1} X[k] e^{j\frac{2\pi nk}{N}}$$

A straightforward calculation of the DFT requires N^2 complex operations. By constraining N to be a power of two (for example, 512, 1024, and so on), an algorithm called the fast Fourier transform (FFT) can be used to effectively do the same operation with fewer operations. Depending on the length of the vector, the FFT operation can still be computationally intensive, requiring $(N)(\log_2 N)$ complex operations. Thus, it's important that the implementation in the vector math library be as efficient as possible. In fact, literature for vector math libraries often shows the speed of the FFT as a specification parameter for the product. For the reasons mentioned previously, it should come as no surprise that vector math libraries implement the FFT in ways that take full advantage of the speed and architecture of the underlying hardware. This will often impact the way the application can best utilize the vendor-



Marco Dalla Gasperina,
Senior Software Engineer,
CodeWright aficionado since Version One.
www.inovise.com

CodeWright®

The Programmer's Editing System™

Marco is one of the team of programmers bringing early low cost detection of heart disease to reality. Inovise Medical is developing software that uses waveforms from a standard EKG test to provide a graphical view of the size and location of damage to a patient's heart.

The editor behind the code: CodeWright. When precision counts the most.

Download an evaluation copy at **www.starbase.com/embedded** or call **(800) 491-6784**.

$299
Single User
Multi-User License available
Windows 32-bit platforms

starbase®

Premia, a Starbase company

LISTING 5 SHARC-specific FFT implementation

```
/* sharcplus.c */
#define SH_MAX_FFT 8192
extern float dm seq_dfft[];
extern float pm seq_pfft[];

/* memory required by vendor but invisible to the application are statically
defined */

static int wtablesize = 0;
static float pm *wr;
static float dm *wi;
static float dm *fftdatavec;
static float dm *xr;
static float dm *xi;

void
init_fft_sys(int fft_size)
{
      /* check if exceed architecture's capacity */
      if(fft_size > SH_MAX_FFT)
        return;
      /* Initialize aligned memory pointers */
      /* real ffts */
      wr = seq_pfft + (SH_MAX_FFT/2);
      wi = seq_dfft + (SH_MAX_FFT/2);
      /* for complex ffts */
      xr = fftdatavec;
      xi = fftdatavec + (SH_MAX_FFT/2);
      fftdatavec = seq_dfft + (SH_MAX_FFT/2) + (SH_MAX_FFT);
      /* recompute weights if new maximum is found */
      if(fft_size > wtablesize)
      {
        wtablesize = fft_size;
        fftwts(wr, wi, wtablesize);
      }
}

/* Implementation of library independent API */
void rvfftb_sys(float *a, float *c, long n, long flag)
{
      /* we must provide the twiddle factor
          table stride manually */
      int wstride = 2*(wtablesize/n);
      if(flag==1) /* forward FFT */
      {
        /* Careful, rfft does in-place computation */
        /* Let's copy the input to the mem aligned buffer */
        vmov(a,1,fftdatavec,1,n);
        rfft(fftdatavec, wr, wi, wstride, tempdm, temppm, n);
        vmov(fftdatavec, 1, c, 1, n);

        /* unpack output */
        c[n] = c[1];
        c[n+1] = 0.0;
        c[1] = 0.0;
      }
      else /* inverse FFT */
      {
        /* cffti requires that the input be divided into
        real and imaginary components */

        vmov(a, 2, xr, 1, n/2 + 1);
        vmov(&a[1], 2, xi, 1, n/2 + 1);
        vmov(&xr[n/2 - 1], -1, &xr[n/2 + 1], 1, (n/2 - 1));
        vmov(&xi[n/2 - 1], -1, &xi[n/2 + 1], 1, (n/2 - 1));
        vneg(&xi[n/2 + 1], 1, &xi[n/2 + 1], 1, (n/2 - 1));

        cffti(xr,xi,wr,wi,wstride,tempdm,temppm,n);

        vmov(tempdm,1, c, 1, n);
      }
}
```

specific features of the FFT, especially while maintaining a portable code base.

Vector math libraries often take advantage of the fact that the FFT algorithm relies on a fixed set of weighting coefficients that can be stored once prior to calling the FFT function:

$$W_N^k = e^{-j\frac{2\pi k}{N}} = \cos\left(\frac{2\pi k}{N}\right) - j\sin\left(\frac{2\pi k}{N}\right)$$

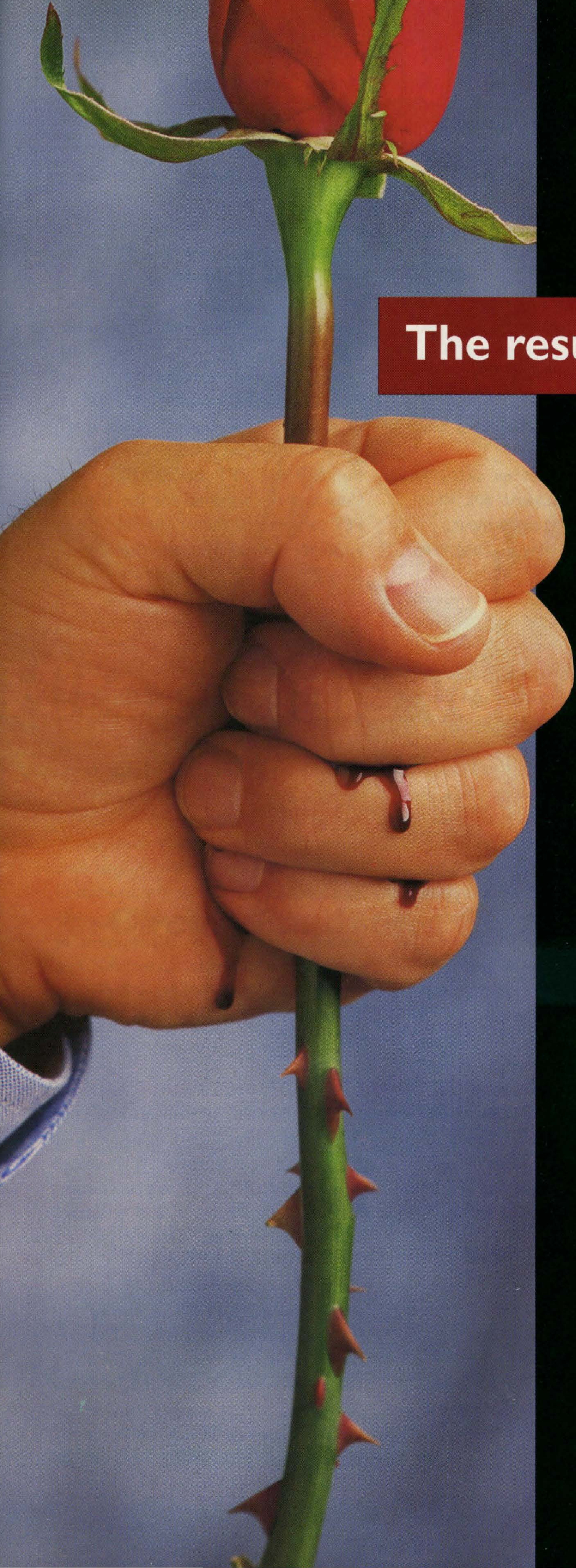
A set of these coefficients (sometimes called twiddle factors) need only be computed once for the longest FFT computation required by the system, since subsets of this complex vector can be used for any shorter FFT vector calls. Vector math libraries sometimes provide an initialization function for creating these weights (see examples in Listing 4). Other libraries require that an initial call to the FFT routine be made to initialize the coefficient table. Since more than one module may need to calculate an FFT, and since the vector sizes may vary within the same application, we also define an API for the FFT initialization function. Not only will this make the software portable across vector math libraries, it can also be used to track the maximum table length required by the system.

In our real-time signal processing software, we defined a function:

```
void init_fft_sys(int size);
```

called at the application level whenever a signal processing module executed setup code prior to computing the FFT. Using either the preprocessor macros or supplemental C code, the `init_fft_sys()` function was made to call the appropriate underlying library initialization method. As we shall see later, this function can also serve to make other required initializations.

Vector math libraries often come with not just one, but a set of FFT routines, each one designed to take



Rational Rose for real-time?

The result could be painful.

More and more development teams agree Real-time Studio® is the better system and software modeling solution.

It makes perfect sense. If you're developing real-time embedded systems, you need a software modeling solution that can support your entire team and will actually fit your existing process. Make no mistake — reach for Real-time Studio. This is the one and only software modeling solution with an object-based repository enabling complete, up-to-date, shared views of your system. Finally, your entire development team can communicate and collaborate — from start to product delivery. Developed exclusively for real-time systems, Real-time Studio is non-intrusive, flexible and features automatic code generation for C, C++, and Java. Today, it's the proven solution for hundreds of developers and development teams. And the list keeps growing. Get started on the fastest path to the right product. Visit **www.artisansw.com/real** for more information and a demo of Real-time Studio, plus you can register for our upcoming Real-time UML seminar.

ARTiSAN SOFTWARE

Real-time Studio®
www.artisansw.com/real

No developer was actually harmed or risked injury in the making of this ad.
Rational Rose is a registered trademark of Rational Software Corporation.
Real-time Studio is a registered trademark of Artisan Software Tools, Inc.

advantage of either some property of the transform or some property of the hardware.

Vector math libraries often differentiate between a "real" FFT and a "complex" FFT. The real (forward) FFT assumes that the input vector, $x[n]$, consists of real values (that is, the imaginary parts are zero). By making this assumption, it takes advantage of the property that the resultant transform, $X[k]$, will consist of N complex quantities related as follows:

$$X[k] = X^* \, [N-k] \quad for\ k = 0,...,N-1$$

where * denotes the complex conjugate. From this it can be seen that the $X[N/2]$ is a real value (since $X[N/2]$ must equal $X^*[N/2]$). It can also be shown that the $X[0]$ term is real valued as well. Knowing this, vector math libraries can save on computation and output storage by calculating only the first $N/2 + 1$ values consisting of $X[0]$ (real), $X[1]$ through $X[N/2\text{-}1]$ (generally complex), and $X[N/2]$ which will also be real. When using the real FFT routines provided by a vendor library, the responsibility is on the application programmer to synthesize the conjugates not provided by the routine. Comprehensive libraries may provide fast routines to expand the output vector to its full N point length, but it is rarely necessary to invoke these since most applications are only interested in the first set of values.

The "complex" FFT operation makes no assumption about the realness of the input vector, $x[n]$ and so no FFT properties are leveraged. The output of N point complex FFT routines will produce N complex values.

A variant of the FFT algorithm that uses in-place computations can reduce the number of complex operations further to $(N/2)(\log_2 N)$. The tradeoff for this reduction in computation is the loss of the input vector data as it is overwritten by the output values. It is sometimes the case that vendor libraries will only provide in-place routines and leave it to the programmer to manage the input and output issues according to the application.

In our real-time signal processing application, APIs for both the real and complex FFT function calls were defined (for legacy reasons) as follows:

```
/* real FFT function call, vendor-
     independent API */
void  rvfftb_sys(float *input,
  float *output, long n, long flag);

/* complex FFT function call,
     vendor-independent API */
void cvfftb_sys(float *input,
  float *output, long n, long flag);
```

The flag variable indicates whether the transform is a forward or inverse operation. The values at the first address represent the input ($x[n]$ or $X[k]$ depending on flag) and the values at the second address represent the output vector. For example, a real N point forward FFT using `rvfftb_sys()` would expect an array of N floats at the input address and would result in $(N/2 + 1)$ complex (float pairs) at the output address. Obviously, these routines are generalized for not-in-place computations; an in-place API could be defined or assumed if the input and output addresses are identical, depending on the underlying implementation.

Many vector math libraries use a concept called packing to represent input and output vectors. Going back to the property of the real valued forward FFT, since the $X[0]$ term and the $X[N/2]$ are guaranteed to be real valued, implementations will combine the elements in the first term of the complex output:

```
/* pseudo code */

float output[N];

output[0] = X[0]
output[1] = X[N/2] /* pack the last
  real value with the first */

output[2] = Re{X[1]}
output[3] = Im{X[1]}
.
.
output[N-2] = Re{X[N/2 + 1]}
output[N-1] = Im{X[N/2 + 1]}
```

This normalizes the sizes of the input and output vectors to be N floats, whether representing the real input values or the $N/2$ complex values.

Another aspect that vector math libraries differ on is the precise definition of the FFT being computed. Consider the formula for the inverse FFT where a factor of $(1/N)$ is applied. Not all implementations of the inverse FFT will apply this factor and must be done so explicitly (using another library call) if it is an implicit assumption about the vendor-independent API.

Probably the most important aspect of defining an API for the FFT (and similar functions) is to document exactly the underlying math that it is expected to perform and how the input and output data are to be presented back and forth to the application. Is it scaled? Is the output data in a packed format? Is the operation in place? Whatever the decision, document it, perhaps as a comment in the application code. Later, this can be compared to new library literature when porting to the next architecture.

FFT example

To conclude this section on FFT considerations, here is an example that takes into account the differences between the `rvfftb_sys()` vendor-independent FFT API and a target-dependent implementation.

The Analog Devices 21060 (SHARC) floating-point DSP processor uses a Harvard architecture memory system. As such, it has two memory banks, program memory and data memory, and it is able to fetch two words during one clock cycle. These attributes are used by one vendor of SHARC libraries to create vector library FFT routines

Don´t gamble with success ...

Hitex deals you winning cards!

USB Agent - USB Analyzer

The USB Agent is a full-featured USB-Protocol analyzing instrument ...
A development tool that captures, analyzes and intelligently displays all USB-signals. Right performance - Right price!

In-Circuit Emulators for ...

MX430 family:

For MSP430 microcontrollers, these emulator systems support all members of this TI family of low-power microprocessors, including the ultra-low-power MSP430x11x derivatives. These instruments are proving to be very popular.

C166 family: **DProbe167 and DBox167**

A modular family of powerful in-circuit emulator systems for the Siemens C16x variants featuring the most advanced adaptation technology: PressOn. Our C161 system with a 64 K trace buffer module is available for less than $3,500. Also supports ST-10 microcontrollers.

68HC12 family: **DProbeHC12**

For 68HC12 processors; supports maximum speed and all operating voltages; plug and play (no jumpers, no switches); additional BDM interface cable; flash programming built-in; PlugOn trace with 32k frames. Move up to super advanced features with DBoxHC12.

8051 family: **MX51/AX51**

True real-time emulation for more than 500 variants from all manufacturers. Including derivatives like Atmel 89S8252, Intel 8x931x and Siemens C505C.

and more:

251, 68k, CPU-32, 80x86, 68HC11, Pentium® Processor

hitex

DEVELOPMENT TOOLS

Hitex USA
710 Lakeway Dr., Suite 280
Sunnyvale, California 94086

	Tel.:	(800) 45-HITEX
	Tel.:	(408) 733-7080
	Fax:	(408) 733-6320
	E-mail	info@hitex.com
Hitex Germany	Tel.:	(0721) 9628-133
	Fax:	(0721) 9628-149
Hitex UK	Tel.:	(01203) 69 20 66
	Fax:	(01203) 69 21 31
Hitex Asia	Tel.:	(65) 74 52 551
	Fax:	(65) 74 54 662

www.hitex.com

Advanced technology that brings results...

"The best emulator that I ever used!"

Benefit from our world-wide support:

Austria Tel: +43-1-259727/20 Belgium Tel: +31-77-3078438 PR China Tel: +86-10-62383376 Czech Republic Tel: +420-2-66316661 Denmark Tel: +45-43-424742 France Tel: +33-1-39611414 India Tel: +91-22-8520817 Israel Tel: +972-3-7657666 Italy Tel: +39-0434-633500 Malaysia Tel: +603-7167591 Netherlands Tel: +31-77-3078438 Pakistan Tel: +92-51-822075 Poland Tel: +48-22-7556983 Russia Tel: +7-812-3252792 Singapore/Indonesia Tel: +65-7496168 South Korea Tel: +82-2-645-0386 Spain/Portugal Tel: +34-91-6401234 Sweden/Norway/Finland Tel: +46-87-404580 Switzerland Tel: +41-1-3086666 Taiwan Tel: +886-2-26983456 Turkey Tel: +90-312-4472700 PBX

When supporting a portability API for several targets, it's useful to have a regression test to verify the operation of the individual vector functions.

that are fast but unusual in their calling structure.

Here is the vendor's API of the vector library FFT routine used for real valued inputs:

```
void rfft(float dm *data,
  float pm *wr, float dm *wi, int wstr,
  float dm *tmpdm, float pm *tmppm,
     int n);
```

The "dm" and "pm" in the parameter list are C language extensions of the SHARC compiler, specifying whether the variable is in data memory or program memory, respectively. To use this routine, the programmer will need to initialize a set of real (wr) and imaginary (wi) weight values using the `fftwts()` initialization routine provided by the vendor. The application designer must determine what the weight stride should be for any particular call. Furthermore, some scratch space proportional to the FFT length needs to be on reserve in both data and program memory. All vectors must also be memory aligned to an *n* word boundary. What is more, the function assumes that the format of the output data array is packed and the algorithm is performed in place. Put together, `rfft()` appears painfully incompatible with the `rvfftb_sys()` function defined for the portability API.

As in other cases, we defined a supplemental module for the target, `sharcplus.c` and `sharcplus.h`, which defines complicated implementations of the API in terms of a vendor or hardware target. The `init_fft_sys()` initialization routine is shown in Listing 5. Note the variables defined statically at the top of the module. These members will help encapsulate the target-specific requirements for program and data memory, as well as help in tracking the maximum FFT size required by the system.

Listing 5 also illustrates how the intricacies of the FFT implementation can be buried in the `init_fft_sys()` and `rvfftb_sys()` function calls used by the portable application code. As long as the universal API is general enough to perform the function in ways the application needs, the vendor-specific calls can be manipulated to produce the expected functionality. Of course, some overhead is involved, but that is sometimes the cost of writing portable software.

Memory alignment

On some platforms, vector addresses must be aligned to a particular word boundary in order to work correctly or efficiently. Vector math libraries that accompany hardware usually have functions to load the data into memory aligned addresses. However, explicitly calling these routines in the application software can clutter the code as well as make it inefficient for platforms that do not require memory alignment.

One solution is to abstract the vector memory allocation routine and constrain the target-dependent implementation to allocating memory aligned to the required boundary condition. This routine can be redefined to return pointers that obey memory alignment considerations for a specific architecture.

Regression testing

When supporting a portability API for several targets, it's useful to have a regression test to verify the operation of the individual vector functions. Defining the regression test is also a good way to make clear the expected inputs and outputs of each function. Fortunately, only one regression test is needed per function because they will be portable to all platforms. The test program should call each function with known values and check the output vectors for the correct data. Because of slight differences in floating-point roundoff, all of the targets might not return bit-for-bit equal answers. However, they should be the same to within some small tolerance. By automating the regression testing, it is easier to catch target-specific errors before they cause problems in the complete system.

Summary

We have shown several techniques for using vector math libraries in a way that is portable to many different libraries. Most of these techniques involve the C preprocessor, which can substitute target-specific code during compilation. Alternatively, function wrappers can be used to encapsulate the target-specific code. Function wrappers tend to be more readable than function macros and are appropriate when the mapping from the portability API to the vendor API is complex. We use function wrappers to encapsulate the FFT functions on most targets. Macro functions, while being more obtuse, do not introduce the overhead associated with a function call. In some performance-critical areas, that overhead can be important. **esp**

James Metzger is a senior engineer at BBN Technologies, a unit of GTE. He has many years of experience in the development of real-time signal processing and display systems, primarily for sonar applications. James received his M.S. in electrical engineering at Johns Hopkins University and a B.S. in the same subject at the University of Maryland. His e-mail address is jmetzger@bbn.com.

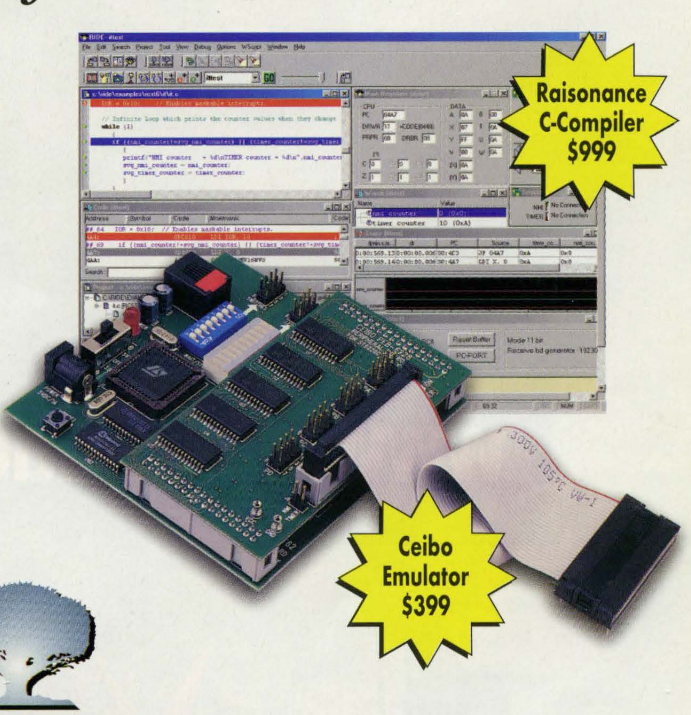
William Wright is also a senior engineer at BBN Technologies. He has 10 years of experience in real-time signal processing systems development and object-oriented design. He received his M.S. in computer science at George Mason University and a Bachelor of Music Education at Indiana University. He can be reached at wwright@bbn.com.

A Golden Opportunity

Get The Winning Development Tools

Best Support for ST6 µCs

New golden tool chain is available from Ceibo and Raisonance for Gold ST62 µCs.
The development tools support most common STMicroelectronics ST62 µCs, all in a box.
The Ceibo EB-ST62 Emulator includes Raisonance Windows IDE and software, which can be easily upgraded to a highly optimizing C-Compiler, the first ever for ST6 family. This is the golden opportunity for design wins and quick time to market with a complete range of 8-bit devices.
For more information, or to place an order, call Ceibo at 1-800-833 4084 or contact us via the Internet at eb-st62@ceibo.com.
Online orders also accepted at www.ceibo.com.


RIDE - #test
File Edit Search Project Tool View Debug Options WScript Window Help
GO

```
IOR = 0x10;    // Enables maskable interrupts.

// Infinite loop which prints the counter values when they change
while (1)
    {
    if ((nmi_counter!=svg_nmi_counter) || (timer_counter!=svg_timer
        {
        printf("NMI counter    = %d\nTIMER counter = %d\n",nmi_counter
        svg_nmi_counter = nmi_counter;
        svg_timer_counter = timer_counter;
        }
```

Raisonance C-Compiler $999

Ceibo Emulator $399

CEIBO
Development Tools of Choice
www.ceibo.com
info@ceibo.com
1-800-833 4084

RAISONANCE
www.raisonance.com

mcu.st.com

Product and company names are trademarks of their respective organizations

Don Morgan

DSP and Motion Control

Motion control is a wide-ranging and complex topic. When I first approached the idea of writing about motion control using DSPs, it seemed fairly straightforward. Having dealt with motion control for almost 20 years, I felt I had the experience, knowledge, and scope to do a decent job. Maybe. Once I started, though, I saw motion control did not fit this neat little package I had anticipated. As a result, I have re-written this column several times, looking for a meaningful entry point to the subject.

In the early '90s, one of the first applications for DSP chips was in motion control. Although this area may not seem to have the sophistication associated with signal analysis or synthesis, it employs many of the same tools and mathematics and has many intriguing features of its own. Not only can DSPs provide extremely facile, precise, and programmable control for any application, but the speed of the DSP and its math engine allows other functions that are usually supported as peripherals to be integrated into the software. This provides an immediate size and cost advantage.

With integration increasing and die sizes dropping, peripherals such as encoder interfaces, PWM generators, multiple synchronous A/Ds, and more are available on the same chip with the DSP. Other functions associated with high-end applications (and expense), such as real-time interpolated sinusoidal encoders, resolvers, and other transducers, can be done in software. This can mean that low-cost encoders/transducers may be used to produce very high resolutions.

With this in mind, I would like to address some of the issues of motion control and control theory in relation to DSP. The truth is that control theory and traditional signal processing are not far apart. The mathematics are the same as what we've been using in our discussions of signal processing, but with a different look. The concerns vary and inevitably involve the time domain. What's more, the processing is more likely to be stochastic than many signal processing applications. Nonetheless, a transfer function remains a transfer function.

Although motion control is a traditional DSP function, it isn't so straight forward as it may seem.

In this column, we will look at popular hardware solutions with an emphasis on DSP and PWM, as a number of the manufacturers make parts specifically designed for this increasingly popular form of motion control. We will look at many of the basic algorithms, PID, the Park and Clarke transformations, sinusoidal encoder interpolation schemes, as well as software resolver conversions, PWM generation, and more. And we will delve more deeply into the concerns of control theory.

Our interest is in algorithms, DSPs, and applications. Clearly we cannot really discuss any aspect of these subjects without some familiarity with the concepts of the technology involved. To do this adequately, however, it is necessary to introduce some terms and concepts so that we can see how they relate to DSP and the design of such systems.

What is a motion control system?

What are the components of a motion control system? Typically, there is a trajectory generator and a drive or amplifier for the axis of motion, which may ultimately be a motor but might also be a valve or anything else that the user must control. The trajectory generator commands the drive to produce a given velocity, position, or torque (it may be many things). The drive sources power to the load in accordance with these commands. If motor controls are this uncomplicated, why DSP?

Let's answer this question with an example. Perhaps the simplest implementation of a motion control system would be a basic positioner. People use their cars as positioners. If you wish to drive yourself to work, you probably don't attempt to go from zero to 60mph in an instant, then from 60mph to zero again when you have arrived. The wear and tear on the vehicle and the passenger would be too great for either to last

The truth is that control theory and traditional signal processing are not far apart. The mathematics are the same as what we've been using in our discussions of signal processing, but with a different look.

long. You must accelerate, that is, increase your velocity, in measured increments until you have reached a speed suitable to you, your vehicle, and the environment. As you approach your destination, you must decide on a point you feel is appropriate to you, your vehicle, and the environment to begin decreasing this speed, so that you can stop in the right place.

The same is true for any motion control system. The function of providing these measured commands to the system is assumed by what is known as the *trajectory generator*, and its functionality can become quite complex, depending upon the needs of the system. Most modern applications have special requirements regarding the equipment they control.

In multi-axis applications, the motion of all the axes must be synchronized with the focus of the motion—say, a circle, square, rectangle, or some other shape—executing complicated shapes. This synchronization can come in many forms; it may be geared off a single axis or sourced from a central point. In this case, the trajectory generator must be capable of coordinating the many axes and shaping the motion through splining or some other mechanism, so that all axes arrive correctly at the same place and that this motion is executed exactly, smoothly, and economically.

DSP also becomes involved in the other area of discussion, the *drive*. The drive is the part that accepts the command—say, a voltage, current, or digital word, usually signifying torque, velocity, or position—from the trajectory generator, and supplies the current, or some other energy, to the load so that it can move. There are many mechanisms for accomplishing this, both analog and digital, and they too are moving to DSP.

Two of the most popular forms of drives are PWM and analog. The analog supplies current to the windings as sinusoids that change frequency and compel the device to move. The PWM drive turns voltage on and off to the windings at a fixed frequency but varying duty cycle, causing the current to the windings to vary in frequency and magnitude with the duty cycle. Both forms have their advantages and drawbacks. DSP can control them both—the analog through DACs and the PWM through pulse generators.

Of the two, PWM is becoming more popular and is now supported directly on many DSP and microcontroller chips.

Trajectory generators and drives are the basic components. Now let's look at some forms of control.

Open loop or closed loop?

Control is often referred to as either open loop or closed loop. In an open loop situation, the trajectory generator supplies a command to the drive, scaled to produce an expected result. Here, the trajectory generator does not examine and correct for any system errors. An example of such a control might be a micro-stepper controller as a positioner. The user will usually enter some defaults for acceleration and velocity and give the controller a position to go to. Using the defaults, the controller will calculate accelerations and peak velocities based on how far it must go and whether there are any other special

As speeds increased and dies shrank, more and more functions were added to the [DSP] chip, including PWM creation, encoder inputs, and ADCs.

requirements for the move. It will then produce commands for the drive that are expected to take the axis to position. No effort is made to correct for errors the axis may have in attaining velocities or intermediate positions. Using some sort of feedback device (in order to determine when the axis has arrived at its final position), the controller drives the stepper until it arrives at position.

In a closed loop control, the errors are literally added into the equation. Scaled values of the error are added to the original command to better fit the command to the load. If the axis is not attaining velocities, the command is increased in proportion to the error so that it will. This is called servo control—it's driven by error. The loop may be closed on any particular feature in the system, whether it is position, velocity, torque, or some alias.

To view these ideas in another way, imagine an open loop system as a simple operational amplifier. Here, a gain is applied to any input. The result is output. As long as both the input and the output (or load) are well known and bounded, this works fine. But, as with an amplifier, if the input exceeds the parameters of the circuit, you can find yourself with a square wave at the output instead of the nice sinusoid you were hoping for. Such a system can be described with a simple multiplier; it is purely feedforward.

Supplying feedback to the amplifier can produce a filter (or oscillator). A portion of the output is added to the input to refine or adjust the response of the system. This is now a servo system; it produces a transfer function with feedback.

As with any system, you may have any combination of feedback and feedforward mechanisms involved. All of this is useful for highly efficient systems because inadequacies in equipment or the load can be overcome by properly modifying the transfer function; this is part of tuning a servo system.

Designing a motion control platform

Some of the most popular motion control applications today involve servo systems and PWM drives. Typically, these controllers will drive anything from AC induction motors (ACIM), permanent magnet synchronous motors (PMSM), brushless DC motors (BDCM), to switched reluctance (SR) motors in industrial applications. Feedback mechanisms are usually encoders, resolvers, or interpolated sinusoidal position transducers.

To get an understanding of the scope of a motion control design, let's look at it using standard parts for the functions. Knowing such a system provides a nice basis for looking at more highly integrated systems on DSP. So let's say that we have a standard DSP, such as a DSP56k, and we wish to build an integrated trajectory generator and PWM drive. (What I'm describing is not the definitive motion control device, but a simple and common implementation.)

We will assume for such a system that the controller is receiving commands from a computer or responding to commands in its own memory. These commands may be position, velocity, or torque commands, and may involve information about other axes and synchronization.

For the system to function, it must have feedback. For our example here, we will assume that it is a simple incremental encoder signal that arrives at a special counter that incorporates some kind of digital filter to remove noise. This gives us position information, but we also need information about the load (motor) in order to control it effectively. This is usually current feedback from two of three axes for a three-phase motor, one of two for a single-phase motor.

Now that the DSP can see position and can compute currents, it is ready to drive the motor. In this case, it needs some mechanism to create fixed frequency pulses that vary in duty cycle according to the values calculated by the DSP. There are a couple of different forms of PWM control, the most popular being center and edge controlled. With edge-controlled PWM, one edge is fixed and the other edge moves with the duty cycle changes. With center-controlled, the center is fixed and both edges move.

Either way, the concept is a simple one: a counter of fixed length driven by a clock. You vary the duty cycle of the output by using different taps to control turn-on and turn-off time. Some manufacturers have rather complex chips for this purpose, but it can be done with a latch if you are willing to sacrifice some computing power and time to the function. And, of course, these pulses must be manufactured for each phase of the motor with sufficient dead time for top and bottom drive transistors to prevent accidental shorts (shoot-through). In a typical three-phase brushless drive, there are six signals—one each for the top and bottom IGBT/FET of the half-bridge associated with that phase. The length of the counter does make a difference; it defines the sort of resolution you can attain. Greater resolution means finer control.

Now you add the motor phase leads to the center of each half-bridge and power and you are ready to go. Of course, we have not looked at the software necessary to drive a three-phase or single-phase PWM system, nor the code for interpreting the current feedback or creating a PID loop or using interpolating sinusoidal encoders, nor resolvers, nor many other things—but we will.

In many modern implementations the trajectory generator and drive are one and the same. You will see how many of the DSPs we present provide functionality for that.

Current integrated DSP offerings

Initially, the DSP was the core of the controller. It was capable of the very fast computation necessary to perform the PID filter operations, position calculations for determining the next velocity. As speeds increased and dies shrank, more and more functions were added to the chip including PWM creation, encoder inputs, and ADCs. Now it's possible to create a complete, high-performance servo application with only a few parts and integrated gate bipolar transistors (IGBTs). This results in inexpensive and interesting applications, such as controller/drive/motor units that may be connected to a central control by merely an optical link or copper pair.

Among the leading contenders for DSP-based motion control are families from Texas Instruments and Analog Devices. Both offer on-board A/D converters, PWM generation capability, counters that can be used for encoder inputs and serial ports for communication, and the addition of external A/Ds and sensors. Both offer a wealth of built-in motion control functions, in addition to standard DSP fare.

TI's offering is the TMS320C24x family. It's a 16-bit, 3.3V, 30 MIPs device with on-board flash and up to 32K words of RAM. Some of which is dual access, allowing more than one access during an instruction cycle. There is a multiplexed (with sample and hold) 10-bit A/D and eight 16-bit PWM channels that allow either edge or centered PWM with programmable dead-band to prevent shoot-through on the IGBTs. It also incorporates an encoder interface and A/Ds. In addition to the standard SCI and SPI ports, some devices even have a CAN port.

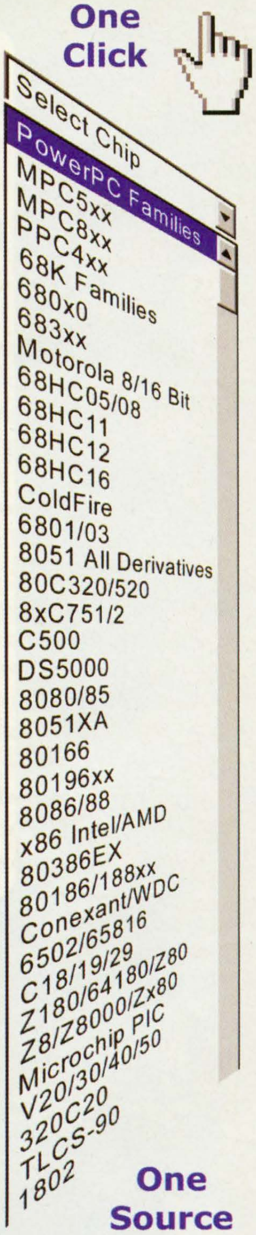
Analog Devices' DSP, the ADMC401, is 16 bits, 5V, and 26 MIPs. It sports a 12-bit flash A/D that permits simultaneous sampling on two channels and multiplexed sampling on eight channels. Again, it supports different modes for its PWM generator and deadtime. There is also a high bandwidth quadrature encoder interface with digital filter and a number of other additives. One strong attraction for the Analog Devices part is that it comes with many of the common algorithms for motion control implemented in masked ROM. This, of course, will reduce time to market and memory requirements.

In many modern implementations the trajectory generator and drive are one and the same. You will see how many of the DSPs we present provide functionality for that. The sort of integration provided by the chips mentioned above is making it possible for the motor, drive, and trajectory generator to be a single unit that needs only power and some form of communications, such as an optical cable or twisted pair.

Next month

At least we have made a start. Next month, we'll start looking at some of the algorithms in common use in today's motion controllers. **esp**

Don Morgan is senior engineer at Ultra Stereo Labs and a consultant with 25 years experience in signal processing, embedded systems, hardware, and software. Morgan recently completed a book about numerical methods, featuring multi-rate signal processing and wavelets, called Numerical Methods for DSP Systems in C. *He is also the author of* Practical DSP Modeling, Techniques, and Programming in C, *published by John Wiley & Sons, and* Numerical Methods for Embedded Systems *from M&T.*



Tools for Embedded Developers

Software

Integrated development environment

The MULTI 2000 IDE v. 3.0 is an integrated development environment that automates all aspects of embedded software development. It features an integrated programming editor, source-level debugger, graphical program builder, run-time error checker, a version control system, instruction set simulator, performance profiler, and real-time event analyzer. The debugger's cross-reference browsing capability enables programmers to view variable and function use throughout the source code. It's available for PowerPC and ARM processors, with other processor support planned. MULTI 2000 is available now for $5,900 for Windows and $8,600 for Unix.

Green Hills Software
Santa Barbara, CA
(805) 965-6044
www.ghs.com

Royalty-free OS

On Time RTOS-32 is a royalty-free embedded operating system that implements a Windows NT subset in 16K. It has a scaleable component architecture and only the components actually needed by the application are loaded onto the target. Components include a development system (RTTarget-32), a real-time scheduler (RTKernel-32), an embedded file system (RTFiles-32), and an embedded TCP/IP stack (RTIP-32). Other features of the On Time RTOS-32 include no run-time royalties, full source-code availability, and free technical support. The On Time RTOS-32 starts at $1,700.

On Time
Setauket, NY
(888) 667-8200
www.on-time.com

Excel add-in

Doctor DAQ is an Excel add-in that imports individual samples, large waveforms, or existing worksheet data directly into an Excel worksheet—displaying real-time data—and runs analog and digital I/O subsystems simultaneously. The product employs a graphical interface that requires no programming. The user also has control of various hardware functions, including the number of points to acquire, screen update rate, digitization rate, source or destination worksheet cells, which channels to use, bipolar or unipolar operation, single-ended or differential operation, and start/stop trigger setup (internal or external, on rising or falling edge). Both a demo mode and a lite mode are available for free at the company's Web site. The full version, which supports the hardware's entire feature set, costs $195.

United Electronics Industries Inc.
Watertown, MA
(617) 924-1155 x 229
www.ueidaq.com

DSP development tool

Code Composer Studio is a DSP development software tool with visual linking and profile-based compiling capabilities. It supports Texas Instruments' TMS320C64x and TMS320C55x DSP architectures. Code Composer Studio v. 1.2 features DSP/BIOS II, an enhanced version of the DSP/BIOS real-time kernel. Features of the DSP/BIOS II include a multitasking scheduler that handles communications among tasks, input/output control with streaming functionality and a mechanism for writing device drivers, real-time analysis, and real-time data exchange (a mechanism for the transfer of data between the development system and the host development environment). Code Composer Studio is available now for C5000 and C6000 DSPs, with updates to be sent to registered users. For either platform, the software package is priced at $2,995.

Texas Instruments
Santa Clara, CA
(800) 477-8924 x 4500
www.ti.com

Development software for DSP

Software Development Kit v. 2.0 is an integrated development environment for DSP products. It allows customers to convert C, C++, and MATLAB files into BOPS ManArray assembly code for their DSP applications. Software Development Kit v. 2.0 includes a system simulator; instruction set simulator; compiler for MATLAB and vector library; GNU-C compiler, assembler, linker, and loader; VLIW packer; register allocator; DSP libraries; and BOPS ManArray technical information and coding examples. Order and pricing information is available through the company's Web site.

BOPS Inc.
Palo Alto, CA
(650) 330-8410
www.bops.com

Development software with updated Java generator

Real-time Studio v. 3.1 is a modeling suite of development tools, whose new features include an updated Java code generator, increased scaleability, and the ability to import various model elements from Rational Rose. Ease-of-use modifications include an external class mechanism, grid lines and color added to diagrams, and a more elaborate find tool. Real-time Studio is available now for Windows 95, Windows 98, and Windows NT. A single-user copy costs $4,995.

ARTiSAN Software Tools
Portland, OR
(888) 511-7975
www.artisansw.com

Hardware

Development environment

NET+OS integrates an embedded development environment, a real-time operating system, and NETsilicon's NET+Works architecture. It includes Green Hills Software's MULTI 2000 Integrated Development Environment; ThreadX RTOS from Express Logic; and the NET+Works drivers, protocols and services for the Web, e-mail, FTP, flash, and SNMP applications. This package comes with a NET+Arm development board including 10/100 Ethernet, two serial ports, two parallel ports, RAM, ROM, and a JTAG port. Also included is a debug tool that connects the JTAG port to the parallel port of the development PC and is used for code downloading and debugging. The complete NET+OS/NET+Works development environment is available now for $19,995.

NETsilicon
Waltham, MA
(800) 243-2333
www.netsilicon.com

Evaluation system

EZ-KIT Lite performs initial evaluations of DSP products, in addition to the complete toolset for a given DSP family. It is currently available for Analog Devices' ADSP21160x SHARC family of DSPs, as well as the ADSP-2189 M-Series. The ADSP-2189M EZ-KIT Lite Evaluation system consists of a stand-alone ADSP-2189M DSP board with an evaluation suite of code generation and debug software. It is available now for $295. The EZ-KIT Lite system for the ADSO-211160M Sharc DSP is available for $595.

Analog Devices
Norwood, MA
(800) 262-5643
www.analog.com

Pattern generator

The TLA7PG2 is a multi-channel, programmable pattern generator module designed with sequential control that plugs into Tektronix's TLA 700 logic analyzer. It generates digital stimuli to stimulate infrequently encountered test conditions in hardware design and software program testing. The TLA7PG2 features up to 64 channels per module and supports 268MHz clock rate for data output. It's available now starting at $12,995.

Tektronix
Beaverton, OR
(800) 833-9200
www.tektronix.com

State and timing analysis modules

The Agilent 16750, the Agilent 16751A, and Agilent 16752A allow engineers to capture data on buses operating at claimed clock speeds up to 4000MHZ. They also provide up to 32Mbits of memory depth and 2GHz timing analysis on all channels. The new modules are available now and are priced as follows: Agilent 16750A (4Mbits samples memory) for $22,500, Agilent 16751A

(16Mbits samples memory) for $27,000, and Agilent 16752 (32Mbits samples memory) for $35,000.

Agilent
Santa Clara, CA
(800) 452-4844 x 7023
www.agilent.com

Board support package

The Board Support Package allows users of Wind River Systems' Tornado platform and VxWorks real-time operating system to build and deploy runtime images of the operating system on the Adastra VNS-786 and Neptune EBX-size single board computers. The BSP includes all files and images necessary to configure the Tornado development environment to build VxWorks operating system and application images to run on the VNS-786 and Neptune targets. The BSP also includes a pre-built image, with Ethernet support, that can be used as the initial means to establish communication from the target to the Tornado development system. Bootable from floppy, IDE, or M-Systems DiskOnChip, the pre-built image allows the developer to quickly and easily download and debug new OS images. The Board Support Package is available now for $295.

Adastra Systems
Hayward, CA
(510) 732-6900
www.adastra.com

Chips

MPU with V4 architecture

The MCF5407 integrated processor combines the peripheral set of its predecessor, the MCF5307 device, with V4 microarchitecture. It achieves a 257 Dhrystone performance level at 162MHz. On-chip peripherals include commonly used embedded functions such as DMA and DRAM controllers, timers, serial and parallel communication interfaces, and an I²C controller. The MCF5407 is packaged in a 208-pin plastic QFP and is built in a 0.22-micron QLM technology. Production will take place in the third quarter. The device will initially be offered for commercial temperature operation at 162MHz. It costs $19.95 in 10,000-unit quantities.

Motorola
Austin, TX
(512) 895-6217
www.motorola.com

Flash-based DSP family

DashDSP is a series of flash memory-based embedded digital signal proces-

CMP MEDIA PRODUCT CATALOGS

Complete Embedded Product Information

Available Now!

ARM Development Guide
AMD Fusion E86 Catalog
CAN Solutions Directory
DSP Source Guide
Embedded Internet Source Guide
8051 Product Directory
Flash Memory and Components Source Guide
Hitachi Development Tools Catalog
IP Catalog for System-on-a-Chip Design
MIPS RISC Resource Catalog
Philips 80C51 + XA Development Tools
PowerPC Resource Guide
Siemens Microcontroller and DSP Development Tools Guide
USB, PCI and CompactPCI Connectivity Solutions and PC/104 — PC/104-*Plus* Catalog
VantageISI Partners Catalog
X86 Catalog: including 586 & 686 Derivatives
Windows® CE Product Catalog
Windows® Embedded Catalog

ARM DEVELOPMENT GUIDE 1999-2000
1999 CAN SOLUTIONS
1999 DSP SOURCE GUIDE
Embedded Internet Source Guide 1999
8051 1999 EDITION PRODUCT DIRECTORY
1999 FLASH MEMORY AND COMPONENTS SOURCE GUIDE
HITACHI Hitachi Semiconductor (America) Inc. 2000
Microcontroller Development Tools Catalog
1999 IP Catalog for System-on-a-Chip Design
SoC INTEGRATORS/ CONSULTANTS
MULTIMEDIA
DSP CORES
CPU CORES
DESIGN VERIFICATION
OPERATING SYSTEMS
PHYSICAL LIBRARIES
DATA COMM CORES
ASICs with EMBEDDED MEMORY
MIPS RISC RESOURCE CATALOG
QED Quantum Effect Devices
1999 80C51+ & XA DEVELOPMENT TOOLS CATALOG
PHILIPS
Integrated Systems, Inc.
Vantage ISI Partners Catalog
PowerPC RESOURCE GUIDE
MOTOROLA
USB, PCI and Compact PCI Connectivity Solutions Catalog
PC/104 and PC/104-Plus
WindowsNT Embedded Product Catalog
VenturCom
WINDOWS CE PRODUCT CATALOG
Microsoft
X86 CATALOG
Miller Freeman

3 easy ways to get your FREE product catalogs:

1. pick them up the Embedded Systems Conferences
2. order online through www.embedded.com
3. call 1-800-500-6815 in the U.S., 1-785-841-1631 outside the U.S.

Check out all the product catalogs at www.embedded.com

Additional charge applied to multiple orders

CMP
OEM Group

sors that integrates Analog Devices' ADSP-21xx-compatible 16-bit fixed-point DSP core, a three-phase 16-bit pulse with modulator, analog circuits (including a 10-bit A/D converter), voltage reference, current/voltage sensing circuits, a power-on reset, a serial communications port, and flash memory. The first members of the DashDSP family are available. Pricing for the ADMCF326 AC induction motor and the ADMCF328 brushless DC motor is $10 in 25,000-unit quantities.

Analog Devices
Wilmington, MA
(781) 937-1799
www.analog.com

Communications processors

The CS2000 family of communications processors provides a complete and reconfigurable platform that delivers 24,000 16-bit MOPS (million operations per second); 3,000 16-bit MMACS (million multiply-accumulates per second); and 50 channels of cdma2000 chip-rate processing, geared for use in communications infrastructure. Each product in the CS2000 family features a 32-bit RISC processor, a memory controller, a PCI controller, and a reconfigurable processing fabric, all interconnected with a high-speed system bus. The first product in the CS2000 family, the twelve-tile CS2112, will sample in the third quarter. Pricing will begin at $295 each in 100-unit quantities.

Chameleon Systems Inc.
Sunnyvale, CA
(408) 730-3300
www.chameleonsystems.com

OEMs

Single board computer

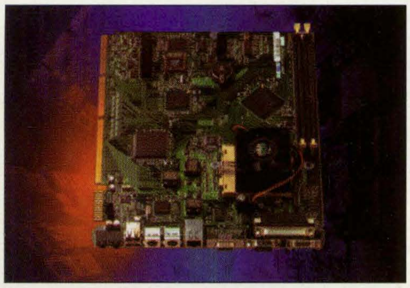
The Elf Single Board Computer is a battery-powered platform designed to address form-factor industrial/militarized instrumentation applications that are serviced through standard peripheral (PCMCIA, IrDA, serial ports, telecom/audio codec) and simple I/O (10-bit A/D converter, general purpose I/O pins). The Elf contains onboard NiMH and NiCAD battery-charging circuitry, self-refreshing, low-power DRAM, standard peripherals, onboard multi-voltage power supply, and direct support for several QVGA and other displays. The price for the ELF SBC is $650.

DVP Inc.
Rockville, MD

(301) 670-9282
www.omnimeter.com

Motherboard

The SF810 micro NLX is a small form-factor motherboard based on the Intel 810 embedded chipset with an integrated VA controller. Expansion cards are mounted horizontally in a riser card. The SF810 features a PanelLink controller for DFP/DVI flat panel monitors, and Intel 82559ER 10/100 Ethernet controller, an AC97 audio, and a PCI/ICA bridge. It has a PGA370 processor socket for use with Intel Celeron and Pentium III processors and two SDRAM sockets supporting a maximum of 512MB SDRAM. It's available now, with prices ranging from $239 to $534.



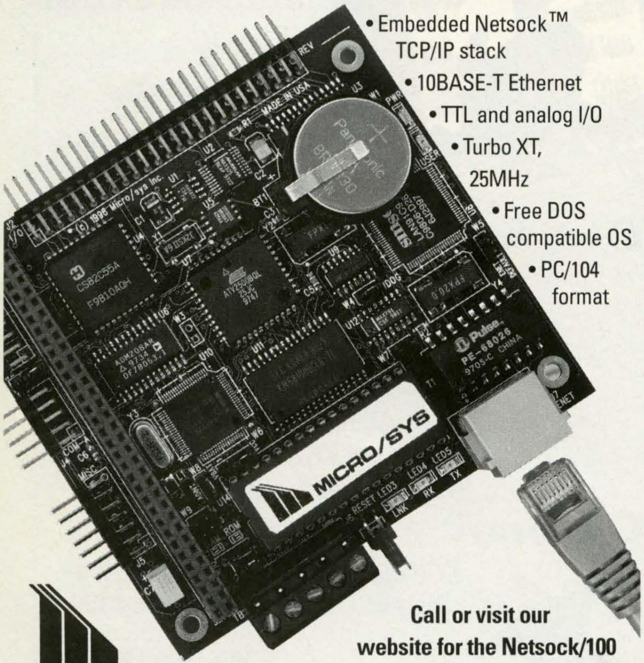
SF810 micro NLX. Intel 310 motherboard from Radisys.

Radysis Corp.
Hillsboro, OR
(503) 615-1100
www.radisys.com

Fast Ethernet controller

The ENET VIII, a PMC card with a Fast Ethernet controller, features eight independent 10/100 Base-Tx Ethernet channels based on Intel's 82559 single-chip MAC/PHY controller. Each channel can operate at 10Mbps or 100Mbps, and each channell supports both half- and full-duplex operation. The ENET VII also includes a digital semiconductor to 21150 PCI-to-PCI bridge, which enables data transfer on the ENET VIII's local PCI bus to occur in parallel with those of a baseboard PCI Local bus. In addition, it provides front-panel access to four Ethernet channels via custom and standard backplanes. It operates from a 5V supply and comes with drivers for Windows NT, VxWorks, and QNX. It available now for $599.

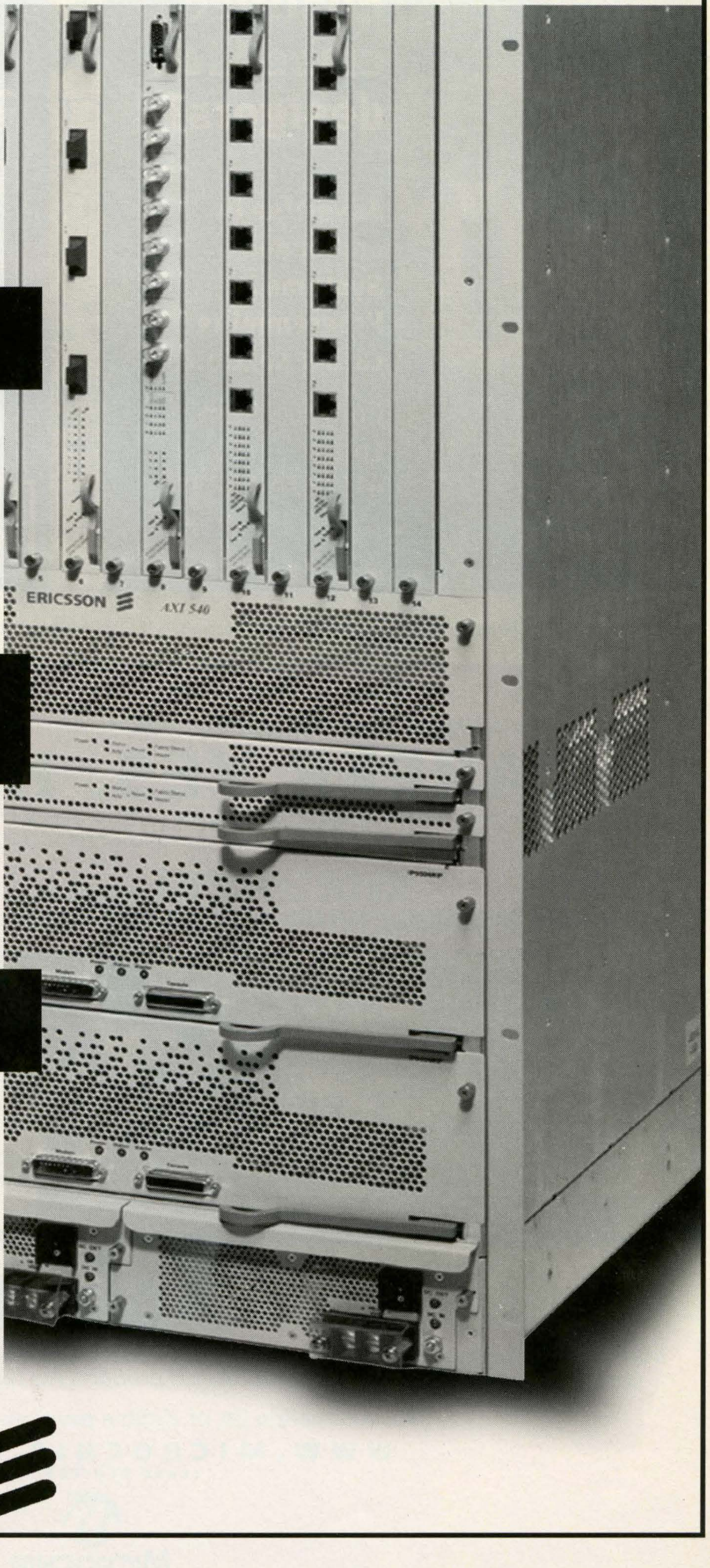
General Micro Systems
Rancho Cucamonga
(909) 980-8463
www.gms4vme.com



Ericsson IP is Redefining the Telecom Industry.

www.ericsson.com/datacom/careers/

- **Senior Interface Software Engineers**
- **Software Tests/SQA Engineers**
- **Senior Digital Design Engineers**
- **Senior Software Engineers/IP Routing Protocols**
- **Interface Software Engineers**
- **Software Engineers/Systems Access Mobile Engineers**
- **Systems Management**
- **Senior ATM Software Development Engineers**
- **ATM Software Development Engineers**



Every day, Ericsson IP is redefining the industry.

As the pacesetting unit within Datacom Networks, Ericsson IP is breaking wire speed barriers every day with the most advanced Gigabit Routers yet made. Our organization represents the absolute cutting edge on the highest level of our high tech industry.

We're focused on going further and faster because we espouse Datacom's mission to lead the evolution of current telecommunications networks. Datacom answers the question of how to have ISP/Data, Wireline, Wireless and Cable through one Multiservice Network. Datacom does it by providing solutions through its end to end product portfolio. And that's where Ericsson IP shines the brightest.

Our AXI 540™ Edge Aggregation Router is the razor's edge.

It's an optimized tool for service provider networks and is designed specifically for the Internet aggregation market. It offers a next-generation, fault-tolerant router architecture and a comprehensive, broadly deployed suite of Internet routing software. The AXI 540™ Edge Aggregation Router features line-speed routing, high-density traffic aggregation, and advanced traffic management features such as service differentiation, VPN provisioning, and traffic engineering.

A corporate culture as advanced as our products.

Although we are a large, super speed, high tech company we have never given up our small start up company mentality. Our company is imbued with a spirit of employee empowerment and an attitude of positive expectation. Our people expect success and the result is a work environment that fosters the highest level of both personal and corporate achievement.

Ericsson IP has the best of both worlds: a global presence along with the distinct feel of family. We're proud of the fact that more than 10% of our top level engineers are women.

As we tell our staff when they first join us, when you work with the very latest technologies, you will truly sharpen your career on today's cutting edge. If you are up to the challenge, please rush us your resume (indicating position of interest). Email address jobs@torrentnet.com ASCII text format preferred. Fax 301-586-8111. EOE. Resumes are treated as confidential.

ERICSSON

ADVERTISER INDEX

Advertiser	URL	Page
Accelerated Technology	www.antinucleus.com	39
Advanced Transdata	www.adv-transdata.com	141
Advantech	www.advantec.com/epc	149
Advin Systems	www.advin.com	143
Agilent Technologies	www.agilent.com	27
Agilent Technologies	www.agilent.com	99
American Arium	www.arium.com	C3
ARC Cores LTD.	www.risccores.com	93
Archimedes Software	www.archimedessoftware.com	82
ARTiSAN Software Tools	www.artisansw.com	119
Avocet Systems Inc.	www.avocetsystems.com	129
Avocet Systems Inc.	www.avocetsystems.com	C4
Axiom Technology	www.axiomtek.com	38
Blackhawk	www.blackhawk-dsp.com	116
Blunk Microsystems	www.blunkmicro.com	142
CAD-UL	www.cadul.com	36
Ceibo	www.ceibo.com	123
CMX Systems Inc.	www.cmx.com	151
Cogent Computer Systems	www.cogcomp.com	136
CONITEC Datasystems Inc.	www.conitec.com	143
COSMIC Software	www.cosmic-software.com	152
Dinkumware LTD.	www.dinkumware.com	135
Domain Technologies	www.domaintec.com	142
EBS	www.etcbin.com	135
EBSnet Inc.	www.etcbin.com	45
Electronic Data Systems	www.eds.com/careers	139
Electronic Engineering Tools	www.eetools.com	141
EMAC Inc.	www.emacinc.com	126
Embedded Power Co.	www.embeddedpower.com	37
Embedded Power Co.	www.embeddedpower.com	83
Embedded Systems Academy	www.esacademy.com	103
emWare	www.emware.com	73
Enea OSE Systems	www.enea.com	29
Ericsson IP	www.ericsson.com/datacom/careers	137
Esmertec	www.esmertec.com	63
Espial Group	www.espial.com	59
EST Corp.	www.estc.com	1
Express Logic	www.expresslogic.com	79
Franklin Electronic Publishers		14
General Software	www.gensw.com	102
Grammar Engine Inc.	www.gei.com	140
Green Hills Software	www.ghs.com	6
Hitachi Semiconductor Corp.	www.superh.com	10
HI-TECH Software	www.htsoft.com	111
HI-TECH Software	www.htsoft.com/pic	142
Hitex Development Tools	www.hitex.com	121
Hitex Development Tools	www.hitex.com	141
Hiware	www.hiware.com	133
IAR Systems	www.iar.com	107
ICE Technology Inc.	www.icetech.com	140
i-LOGIX	www.ilogix.com	97
Infineon Technologies	www.infineon.com	23
Insignia Solutions	www.insignia.com	67
InterNiche Technologies Inc.	www.iniche.com	71
Introl Corp.	www.introl.com	141
iSYSTEM USA	www.isystem.com	140
JK Microsystems	www.jkmicro.com/uflash	142
JK Microsystems	www.jkmicro.com	142
KADAK Products Ltd.	www.kadak.com	77
Keil Software	www.keil.com	86
Lantronix	www.lantronix.com	33AB
Lauterbach	www.lauterbach.com	25
Lineo	www.lineo.com	51
LynuxWorks	www.lynuxworks.com	53
Mentor Graphics	www.mentor.com	95
Metalink Corp.	www.metaice.com	141
MetaWare Inc.	www.metaware.com	113
Metrowerks	www.metrowerks.com	115
Micro Digital	www.smxinfo.com	31
Micro/sys	www.embeddedsys.com	136
Microchip	www.microchip.com	138
Microsoft Inc.	www.microsoft.com/embedded/	75
Microtek International	www.microtekintl.com	4
Microware Systems Corp.	www.microware.com	65
Motorola	www.digitaldna.motorola.com	13
National Semiconductor	www.national.com	89
National Sengineer Search	www.nesnet.com	139
Needhams Electronics	www.needhams.com	140
Nohau	www.nohau.com	C2
Noral Micrologics Inc.	www.noral.com	143
Nucleus Electronics	www.nucleus1.com	143
Pacific Softworks	www.pacsoft.com	47
Paradigm	www.devtools.com	34
ParaSoft Corp.	www.parasoft.com	109
Phar lap Software Inc	www.pharlap.com	76
Phyton Inc.	www.phyton.com	141
PointBase	www.pointbase.com	56
Premia Corp.	www.premia.com	117
QNX Software Systems	www.qnx.com	55
Questlink	www.questlink.com	85
Rabbit Semiconductor	www.rabbitsemiconductor.com	16
RadiSys Corp.	www.radisys.com/SS7	130
Reasoning	www.reasoning.com	35
RLC Enterprises	www.rlc.com	140
Scientific Placement	www.scientific.com	139
Scott Edwards Electronics	www.seetron.com	143
Signum Systems	www.signum.com	140
Structured Tech. Group Inc.	www.stgcase.com	132
Swell Software	www.swellsoftware.com	54
TASKING	www.tasking.com	17
Tech Tools	www.tech-tools.com	142
Tektronix	www.tektronix.com/TLA600	101
Tern Inc.	www.tern.com	140
Timesys Corp.	www.timesys.com	48
Treck Inc.	www.treck.com	43
Trillium Digital Systems	www.trilluim.com	68
U.S. Software	www.ussw.com	40
VerticalNet	www.embeddedtechnology.com	124
Vesta Technology	www.sbc2000.com	141
webPRN.com	www.webPRN.com	87
Wind River Systems	www.wrs.com	8
Working Engines Inc.	www.workingEngines.com	139
WSI	www.waferscale.com	112
Xilinx	www.xilinx.com	81
ZF Linux Devices	www.zflinux.com	8
Z World	www.zworld.com	140

FREE SUBSCRIPTION REQUEST FORM

Embedded Systems PROGRAMMING

P.O. BOX 3404 • NORTHBROOK, IL 60065-9468
www.embedded.com

PLEASE ANSWER ALL QUESTIONS
(FRONT AND BACK) THEN SIGN AND DATE THE CARD.
Incomplete cards cannot be processed or acknowledged.

1. Do you wish to receive/continue to receive EMBEDDED SYSTEMS PROGRAMMING?

□ YES □ No

Signature (required) X________________

Name (please print) ________________

Date ________________

Job Title ________________

Company Name ________________

Address ________________ □ Work □ Home

Mail Stop________________

City ________________ State ________ Zip ________

Phone ( ) ________________

Fax: ( ) ________________

E-Mail:________________

If you do not wish to receive future communications from *CMP/Miller Freeman, Inc.* via e-mail please check here □

If you would prefer delivery to your home, please complete home address. Company name and address are still required to qualify.

Address ________________

City ________________ State ________ Zip ________

2. Check all of the following that you are involved in doing or managing at your company or at those companies to whom you consult. (Please check all that apply)

01 □ Architecture Selection/ Specification
02 □ Writing Software for Embedded Systems
03 □ Writing/Embedding Real-Time Operating System/Kernel
04 □ Debugging Software
05 □ Debugging Hardware
06 □ Hardware/Software Integration
07 □ Hardware/Software Co-Design
08 □ Device Programming
09 □ Project Management
11 □ Software Design/Analysis
12 □ Prototype Testing
13 □ Designing Hardware for Embedded Systems
14 □ Board Layout/Design
17 □ Hardware/Software Co-Verification
18 □ Hardware/Software Partitioning
19 □ Software Testing
20 □ SOC (System-on-Chip) Design
21 □ Internet Appliance Design
15 □ Other (please specify) ________________
16 □ I'm not involved in Embedded Development in any way.

www.espmag.com

B006

3. What is your principal job function? (Please check only one)

Engineering/Computer Management

01 □ Executive Management (i.e., President, VP, Owner, Chairman, Partner)
02 □ Engineering Management (i.e., Technical Director, Chief Engineer, Department Manager, Group Manager)
03 □ Software Engineering/ Programming/Development Management (i.e., Sr Software Engineer, Principal Software Programmer)
04 □ Systems Engineering/ Development Management (i.e., Sr Design, Hardware, or Test Engineer)
05 □ Other Management (please specify)________________

Engineering/Programming Personnel

06 □ Software Engineering/ Programming/Development
07 □ Systems Engineering/ Development (i.e., Design, Hardware, or Test Engineer)
08 □ Engineering Support (Technician, Programming Staff)
09 □ Scientific/R&D/Education
10 □ Other staff (please specify)________________

4. Please check all products which you specify, recommend, authorize, or purchase. (Please check all that apply)

ICs and Semiconductors

01 □ Microcontrollers/ Microprocessors
02 □ 4/8-bit µC/µP
03 □ 16-bit µC/µP
04 □ 32-bit µC/µP
05 □ 64-bit µC/µP
06 □ X-86/Pentium
07 □ Digital Signal Processors
08 □ EPROM / EEPROM
09 □ Flash
10 □ DRAM/SRAM
11 □ Communication ICs
12 □ Media Processors
13 □ CPLDs/FPGAs
14 □ System-on-Chip (SOC)
15 □ MCU Peripheral Chips
16 □ Hardware IP/Cores

System Boards

20 □ Single Board Computers
21 □ VME Boards
22 □ Embedded PCs
23 □ PCI Boards
24 □ cPCI Boards
25 □ DSP Boards

Computer Systems

29 □ PCs
30 □ NT Workstations
31 □ Unix Workstations
32 □ Linux Workstations

Software

36 □ Real-Time Operating Systems/ Kernels
37 □ Compilers/Cross Compilers
38 □ Assemblers/ Cross Assemblers
39 □ Software Debuggers
40 □ Object-Oriented Design Tools
41 □ Simulators/Modeling Tools
42 □ Version/Change Control Software
43 □ Communications Software/ Protocols
44 □ ROMable DOS Tools
45 □ Device Driver Tools
46 □ Embedded Databases
47 □ Embedded Web/Internet Tools
48 □ GUI Development Tools
49 □ Open Source Tools
50 □ Java Tools
51 □ Software Testing Tools
52 □ Integrated Development Environments (IDEs)
53 □ Windows CE Tools

Design Tools/Test Equipment

56 □ In-Circuit Emulators
57 □ Logic Analyzers
58 □ Oscilloscopes
59 □ Data Acquisition Equipment
60 □ Device Programmers
61 □ Hardware/Software Co-Design Tools
62 □ Hardware/Software Co-Verification Tools
63 □ None of the above

5. What is the primary end product or service performed at your location? (Please check only one)

01 □ Computers/Peripherals/Office Automation
02 □ Communications/Telecommunications/Networking
03 □ Consumer Electronics/Entertainment/Multimedia
04 □ Automotive Transportation Systems and Equipment
05 □ Government/Military Electronics
06 □ Aerospace/Space Electronics
07 □ Industrial Controls
08 □ Electronic Instruments/ATE/Design & Test Equipment
09 □ Medical Electronic Equipment
10 □ Other: (please specify)________________

6. What is your engineering/development responsibility? (Please check only one)

01 □ I manage an engineering or software development department
02 □ I manage a project team
03 □ I manage a project
04 □ I am a member of a project team
05 □ Other (please specify)________________

MORE QUESTIONS ON BACK →

CMP

7. Do you specify or buy through distributors?

01 □ Yes
02 □ No

8. How many employees are there at your company?

01 □ 1000 or more
02 □ 500-999
03 □ 250-499
04 □ 100-249
05 □ 50-99
06 □ 1-49

9. Please check the publications below that you receive personally addressed to you by mail. (Please check all that apply)

02 □ EDN
03 □ EE Times
04 □ Electronic Design
06 □ Penton's Embedded Systems Development
05 □ None of the above

10. How many other people read your copy of EMBEDDED SYSTEMS PROGRAMMING?

01 □ 1
02 □ 2
03 □ 3
04 □ 4
05 □ 5
06 □ 6
07 □ 7
08 □ None

11. Please list the names of others at your location who may be interested in a free subscription to EMBEDDED SYSTEMS PROGRAMMING

Name Title

Company name

Company address

City State Zip

PUBLISHER RESERVES THE RIGHT TO SERVE ONLY
THOSE INDIVIDUALS WHO QUALIFY.

www.espmag.com

FOR FASTER SERVICE FAX BOTH SIDES TO: (847) 291-4816

FOLD FOR MAILING

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 2253 NORTHBROOK IL

POSTAGE WILL BE PAID BY ADDRESSEE

CMP

EmbeddedSystems
PROGRAMMING

PO BOX 3404
NORTHBROOK IL 60065-9468

PLEASE TAPE ALONG THIS EDGE. NO STAPLES.

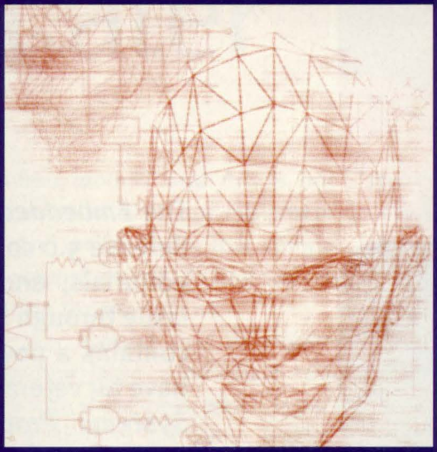
ICSPAT

International Conference on Signal Processing Applications & Technology

ICSPAT

October 16-19, 2000 • adam's Mark hotel • dallas, tX, usa

Meeting of the Minds



View the complete **ICSPAT** program online at **www.dspworld.com**

4 easy ways to register

Web: www.dspworld.com
Phone: (800) 789-2223
Fax: (415) 278-5390
Mail: ICSPAT
C/o Corporate Registration
P.O. Box 612768
Dallas, TX 75261-2768

www.dspworld.com

attention signal processing engineers and programmers!

Don't miss this unique opportunity to enhance your signal processing knowledge and design skills. Take advantage of the many opportunities available this fall to enhance and update your signal processing knowledge at **ICSPAT**, the premier international forum for emerging signal processing research and development.

At **ICSPAT**, you can review cutting-edge research from around the world. Hundreds of white papers will be presented in more than 30 application areas including: Audio, Biomedical, Communications, Industrial Applications, Image Processing, Multimedia, Robotics, Telephony and more!

What you'll get at icspat:

- Access to white papers in more than 30 application areas in the ICSPAT program
- A window on the future of signal processing and the applications in which it is used
- High quality technical knowledge and skill enhancement
- Expanded lectures offering more in-depth, enhanced coverage of topics
- Tutorial and plenary sessions as well as free product training
- A chance to network with engineers and researchers from around the world
- Industry activities including the Vendor Seminar Series with leading industry representatives
- The official proceedings on CD-ROM

For a complete listing of **ICSPAT** lecture and poster abstracts, plus a comprehensive guide to plenary sessions, special events and industry activities, go to **www.dspworld.com**.

ESP1

Sponsors:

CMP EETIMES

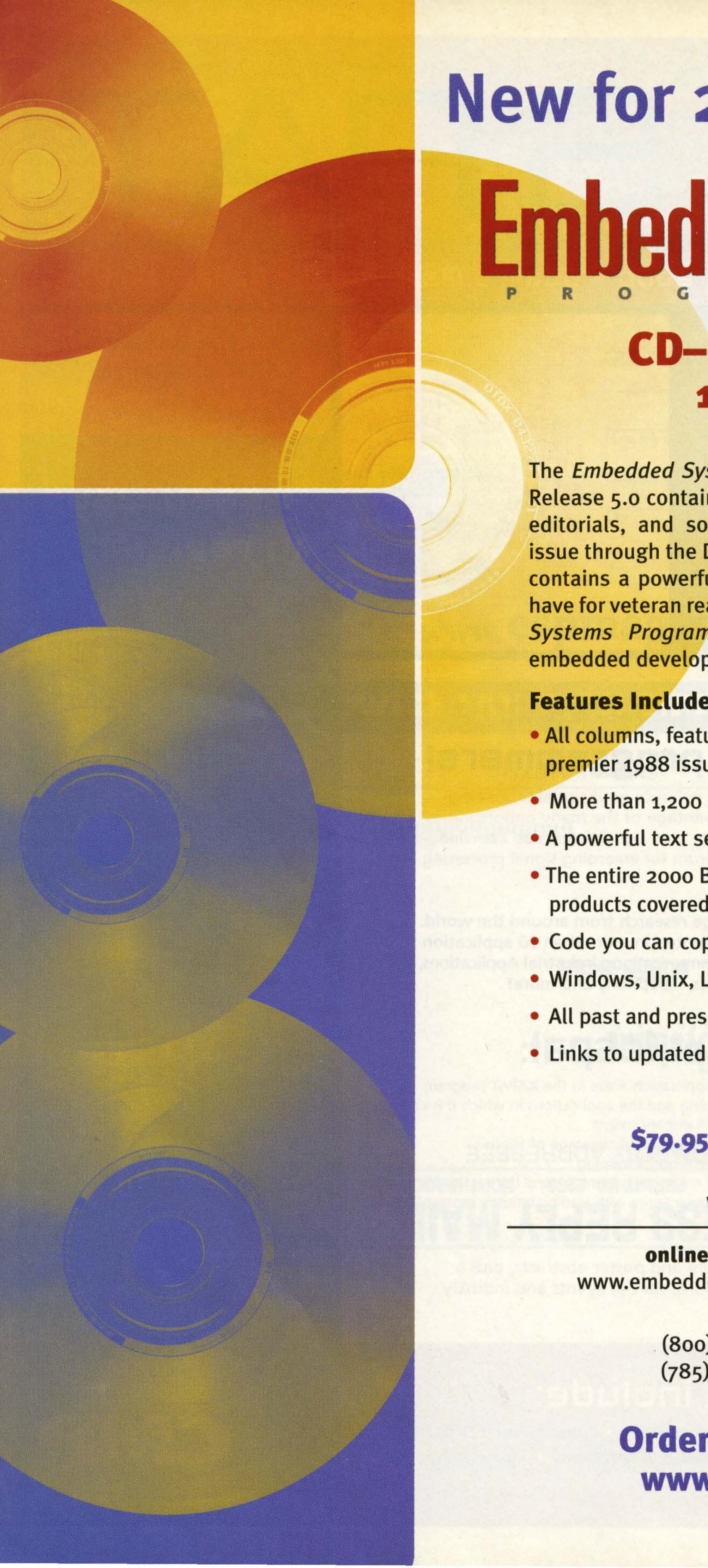
CMP Communication Systems design

CMP EmbeddedSystems PROGRAMMING

Special events include:

• 3 Full-Day Tutorials • FREE Product Training • Plenary Panel Discussion
• Special Guest Lecturers • ICSPAT Awards Ceremony • Keynote Address

CMP



New for 2000...

EmbeddedSystems PROGRAMMING

CD–ROM Library 1988–1999

The *Embedded Systems Programming* CD-ROM Library Release 5.0 contains all columns, features, news items, editorials, and source code from the 1988 premiere issue through the December 1999 issue. This time-saver contains a powerful text search engine and is a must-have for veteran readers and for those new to *Embedded Systems Programming*, the preeminent source of embedded development for more than 11 years.

Features Include:

- All columns, features, and source code from the premier 1988 issue through the December 1999 issue
- More than 1,200 articles—all columns and features
- A powerful text search engine
- The entire 2000 Buyer's Guide—more than 1,200 products covered in detail
- Code you can copy directly into your designs
- Windows, Unix, Linux and Mac compatibility
- All past and present author biographies
- Links to updated information on www.embedded.com

$79.95 new $29.95 upgrade

ways to order:

online
www.embedded.com

e-mail
orders@mfi.com

phone
(800) 444-4881 U.S./Canada
(785) 841-1631 other countries

Order it Online today!
www.embedded.com

Jack G. Ganssle

Ten Years Later

June 1990. DSPs were a rarity. Embedded 32-bit processors were virtually unheard of. C++ was an idea rather than a product. In fact, even C was only just coming into its own as a standard language for building embedded systems. Address spaces were small, as was the firmware. Products were all far more power-hungry than today.

June 1990 was also the first time this column appeared in *Embedded Systems Programming.* Now, 120 columns, a decade, and too much gray hair later, I have to admit to having learned a lot from the readers of *ESP* magazine and my "Break Points" column.

In the last decade readers have sent me literally thousands of e-mails, which, without exception, have been thoughtful and interesting. Even when we have disagreements, no one sends ugly flames; instead there's a thoughtful and polite attempt to change my errant thinking.

And we do have disagreements. I've learned that using the M-word (Microsoft) without immediately following it with "stinks" (or something stronger) riles folks faster than any other subject. The level of passion takes my breath away.

I've learned that open source is not a fad. When I write about how the open source movement bothers my capitalistic sense—that making money is a good thing—readers respond with compelling reasons why the market may be changing. So compelling, in fact, that I really don't have good arguments anymore against open source. Clearly, the market demands new code distribution models, and this is one that seems to satisfy many customers' needs. And we're seeing that companies are smart enough to find ways to profit from the movement.

I've also learned that developers have, by and large, a hate-hate relationship with their tools, which is part of the motivation for open source. When pressed, many engineers will sing the praises of a particular compiler or debugger, but the general dissatisfaction level with tools is truly scary. Bugs, poor support, code bloat, and a wealth of other problems infuriate the embedded community. Vendors ignore this at their peril.

Everything has changed. And yet much remains the same. Jack reflects on 10 years of his column and the changes he's seen.

While most computer users find the GUIs of today a refreshing improvement over command-line interfaces, many developers still find any GUI distasteful. I disagree, since running a half dozen open windows is quite efficient, but I have to respect the large mass of readers who long for the simpler and perceived more powerful days of DOS. Though this is a point of disagreement, we do come together in yearning for tools that easily support automatic operation, rather like the batch files of yesteryear. It's painful to click through a lot of menus to do any repetitive operation.

I've learned that eight bits will never go away. Ever. Analysts and pundits have told me they see eight and 16 bits disappearing over the next year or two, but developers disagree. I'm convinced we're on the brink of an explosion in embedded systems, with embedded processing filling every conceivable niche in our lives. Some of this will be the Internet appliances whose hype saturates all media channels. Much more will be tiny bits of processing, from smart tools to clever pens and intelligent wires. None of these needs a 32-bit monster.

When I look at how embedded design is changing I see some intriguing technologies that were either not available a few years ago, or were crippled by cost issues. For instance, IP cores, though not at all cheap, are now available in a wide array of configurations. FPGAs surround standard architecture CPUs. DSPs greatly reduce the amount of analog electronics required in many applications.

On top of these technology trends, firmware is growing ever bigger, requiring what would once have been considered vast arrays of memory. Time-to-market windows continue to shrink, which is awfully scary when coupled with increased code size.

Perhaps the embedded world is headed in a couple of directions

July 11–12, 2000 • Hynes Convention Center • Boston, MA

Bring this ad with you or register today to avoid on-site lines!

www.embedded.com/summer

EMBEDDED SYSTEMS CONFERENCE SUMMER

$35 Value!

free pass

Bring this ad with you to see:

Exhibit Hours

Tuesday, July 11
12:00 pm – 7:00 pm

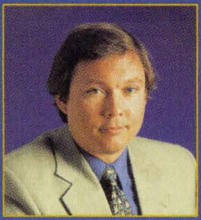
Wednesday, July 12
10:00 am – 4:00 pm

New Expanded Show Floor

ESC Summer's show floor puts you directly in front of today's hottest embedded products. The casual product exhibition with 60+ companies allows more opportunities for one-on-one discussions with leading vendors. Plus, you'll have two days to see the exhibits –1 1/2 more days than in '99. Expect straightforward information in a low-hype, high-tech atmosphere.

Join us and you'll see:

- Embedded Internet Tools
- RTOSes
- Java for Embedded Systems
- MCUs/MPUs
- DSPs
- System-on-a-Chip Technologies
- Development Tools
- Emulators/Simulators



Keynote Address

Jim Turley
"Reformat the Dog:
The World of Weird Embedded Systems"
Tuesday, July 11 • 10:30 am — 11:30 am

Jim Turley, author of *PC's Made Easy* and *Advanced 386 Programming Techniques* asks, *"Where do you want your embedded system to go today? How about up the digestive tract of a senator? Or to Mars, currently the farthest reach of the PowerPC? Do you know about the Internet-enabled lawn sprinkler or the electronic musical lollipop?"* A mixture of technology analysis and plain old weird, Turley's talk includes the strangest, oddest, most fascinating, and most bizarre uses you've ever imagined for embedded microprocessors. It should stimulate questions and raise an eyebrow or two. Turley is vice president of marketing for ARC Cores, Inc., in addition to being an editor, speaker, and industry analyst specializing in microprocessors for handheld, portable, and embedded applications.

Show Floor Reception

"Summer Salsa"
Tuesday, July 11 • 5:30 pm – 7:00 pm

Join us on the show floor to kick back, network with other embedded engineers, and visit the exhibits. This is a great opportunity to talk to the vendors in relaxed mode after they've rolled up their sleeves. ESC Summer sizzles when we serve up margaritas, cervezas, soft drinks, and south-of-the-border treats on this hot! hot! hot! Summer Salsa night. Past attendees say they've made some of their best deals during this reception, so get ready to feel the heat!

ess

Sponsored by:

EmbeddedSystems PROGRAMMING   EETIMES   CMP simultaneously. At the risk of hazarding some prognostications, I suspect that 16 bits may be a market segment that won't survive. Eight bits is cheaper and typically suffers only a 50% performance penalty. Applications needing the bigger address spaces will probably migrate to 32 bits or 32-bit hermaphrodites that use a word-sized bus but run internally with 32 bits. The cost differential is rapidly disappearing.

Eight-bit applications will probably continue to dominate ultra-low-cost and limited power applications, as well as the middle ground of lower volume moderate cost products. A lot of off-the-shelf 8-bit uPs and uCs are perfect for these sorts of products.

But I see 32 bits attacking the high-volume, medium- to low-cost product segment that eight bits traditionally enjoyed. It's pretty expensive to put a big CPU into a small system, but if you're building millions of something, dumping a 32-bit core into a custom ASIC starts to look pretty attractive. As memories become more common on custom chips the total system cost argument will favor bigger processors.

The harbinger of this trend is the automotive industry, a business where engineering even a foot of wire out of a car is a big deal, given the enormous volumes. Even in this traditionally cost-sensitive area, 32-bit CPUs are clearly the future.

A couple of years ago in these pages, I predicted that GUIs would become very common in a wide range of embedded systems, from lawn sprinkler controllers to TV remote controls. Hasn't happened yet, but I still believe this is inevitable. Customers will demand it; we already see GUIs or GUI-like features on sub-$100 products like GPSes. It's pretty clear, though, that embedded GUIs won't run on 8-bit processors, since, to my knowledge, no vendor offers a graphical display for less than 32-bit CPUs. So these high-volume, low-cost apps will require bigger processors.

Different times, same tune

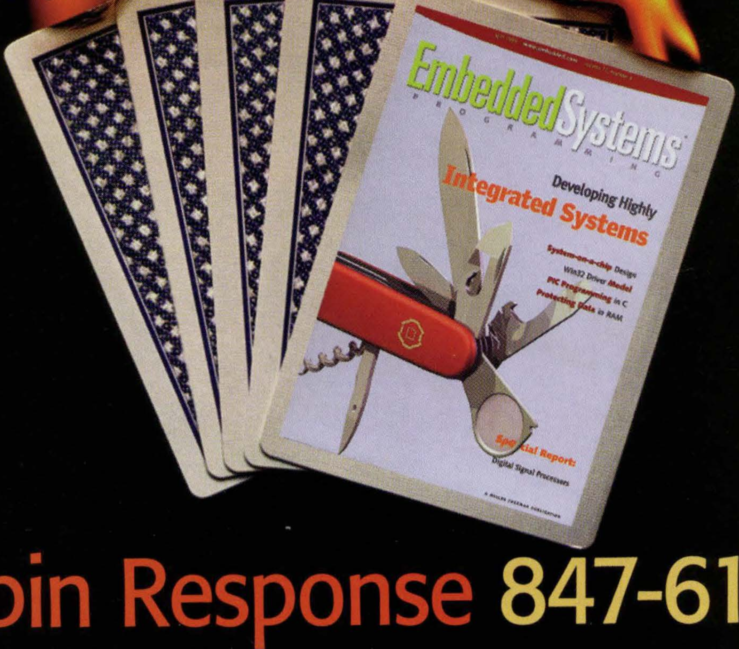
The pace of change in this industry leaves me dizzy. It's impossible to stay current without devoting far too many hours per month reading the trade publications. Yet, despite this seeming pell-mell charge into the future, much remains the same.

We have so many new hardware technologies available today. Hardware design tools have evolved in sync, as well; today's EDA environment could not have been imagined a decade ago. In fact, back in the early '80s most hardware designers created schematics with pencils on huge sheets of vellum. How quaint!

Oddly, the state of software engineering seems little changed from even forty years ago. In 1990 we created designs using (maybe) state charts or data flow diagrams—just like today. Our basic tools were a text editor (as now, with few if any changes), compilers that haven't evolved much, and debuggers which, if anything, are becoming simpler and less capable.

While it's true that the Unified Modeling Language (UML) came into existence in the last decade, the fact is that very few firmware folks use UML, or even have much insight into what it is.

I've learned that reuse has, so far, failed to fulfill its promise. I remember a decade ago lots of enthusiasm for the "software IC," a wealth of reusable software components that would be as common as digital ICs. Hasn't happened, and even the dream seems dead. We all learned that reuse is very hard. Studies indicate that reusable components cost 30% to 200% more than their single-function counterparts. It's awfully hard to invest in the future when the boss is hounding us to ship today.


EmbeddedSystems
PROGRAMMING
Developing Highly
Integrated Systems
System-on-a-chip Design
Win32 Driver Model
PIC Programming in C
Protecting Data in RAM
Special Report:
Digital Signal Processors

The NIH ("not invented here") syndrome is as persistent today as a decade ago. NIH is a prime enemy of reuse. I find developers genuinely enjoy writing code, so resisting the urge to build it yourself is tough. Ten years ago, some 80% of all RTOSes were homemade. Today, for very good and very bad reasons, the figure has slipped just a bit, to perhaps 70%. NIH wins and reuse suffers when one of the only commercially available "software ICs" gains such little mind-share.[1]

Fear of the unknown is a reasonable motivation for building your own OSes and such. Today, more than ever, developers worry that off-the-shelf products may have latent bugs lurking. Never have so many safety-critical applications depended on firmware. Something has to happen to make these software components provably reliable. Perhaps certification to FDA and FAA standards will help.

Maybe we'll see more of these products going the open source route. I can't help but think that Wind River, now the overwhelmingly dominant force in RTOSes, could help themselves and the industry by adopting an open source model for VxWorks and follow-on operating systems. With so few competitive threats there's seemingly little downside. An open source RTOS defeats most of the anti-commercial-RTOS arguments: customers can, if they truly must, maintain the product. They can look under the hood to understand the complex interactions between the OS and their application. And a huge body of very smart people, peering deeply into the code, will both elevate the product's quality and the perception of quality. And as an old friend taught me, perception is reality.

I've learned that despite the passing of 10 years, firmware quality has remained somewhat stagnant. My perception is that many of us are wrestling the quality demon with at best mixed success.

We are accumulating experience: firmware is responsible for quite a few large and small disasters, from the pacemaker that goes awry, to launch vehicle failures, and recently rather dramatic spacecraft failures. As the cost and frequency of these failures multiplies, I suspect customers and governments will demand solutions.

In visiting a lot of companies, I've yet to walk into a firmware shop that's certified to any level of the Capability Maturity Model (CMM). Not that the CMM is a panacea, but it is one of the very few models extant designed to reign in the chaos of software development. Figures show more and more companies becoming CMM-compliant; are these all conventional IT businesses? Why are firmware people so resistant to the idea of adding a software engineering discipline to their processes? Some sort of formalized development strategy is critical to producing high-quality code.

One interesting proposal for the old SDI ("Star Wars") program showed the potential power of reuse to increase quality. The suggestion: require that all of the code has been reused at least three times prior to its incorporation into the SDI project. Avoid the new; recycle old and proven components.

But, it is important for us to recognize our successes at managing quality. What's the difference between a PC and an embedded system? More fundamentally, what's the definition of "embedded system"? In the old days this was an easy question; anything with an 8051 or similar small processor was embedded. Now we see embedded PCs, Linux making inroads into the embedded space, and a variety of other changes that confound a simple definition.

Perhaps the definition of embedded lies in quality. An embedded application runs. Reliably. Rebooting Windows every day or two doesn't affect our lives much. If we had to stop our car every 20 miles to reboot the

engine controller, Detroit would probably go back to mechanical ignitions.

And so I've learned that despite the increasing quality challenges we're facing, so far embedded firmware is, in general, about the only available model of quality code.

Fragments forever

In looking over the embedded marketplace over the last decade, one glaring aspect that remains the same is the horribly fragmented nature of the business.

In June 1990, I was in the tool business—emulators specifically—selling to a tiny fraction of the embedded marketplace. Though we supported a dozen CPUs, most sales calls sounded like, "No, sorry, we don't do that processor." Then, as now, hundreds of different CPUs vied for a share of the embedded space. Each processor requires its own set of compilers, linkers and debuggers, as well as special developer skill sets.

The first couple of decades of this industry saw hundreds of mom-and-pop tool shops spring up. Most failed, a few flourished and grew, the rest grimly held on despite inadequate cash flow. The field is narrower now. Most of those sub-$1 million outfits withered.

A decade ago I thought embedded was the natural realm of small companies serving tiny market niches. Now I see most of the mid-sized business being assimilated, leaving only the gargantuan and miniscule. It's far from clear how this odd mix will serve developers. Or who will survive.

I've also learned that embedded systems live forever. My e-mail inbox fills with stories from readers who are stuck in maintenance on 15-year-old Z80-based products. Or those whose old products have ceased to work because the parts they buy today, with unchanged part numbers, run faster and noisier than their older versions.

Watching the newsgroups (*comp.arch.embedded*), one sees constant postings from developers desperate to get old tools: PL/M, to maintain a product that's two decades old, and the ancient Borland C for an '80s-era x86 product.

It's easy to tell people to check their tools into the version control system, but harder to see how anything will remain intact and available decades hence.

Today's merger and acquisition frenzy means developers often inherit code, sans documentations or tools, while the original engineers gleefully disappear to Tahiti with their newfound IPO millions. So we're still struggling with too much poorly written and inadequately documented firmware.

Finally, I've learned that Tom DeMarco is wrong. He ruefully complains that software folks don't read. I've found they're hungry—desperate—for information. Readers devour this publication, and write when something is incorrect or not clear.

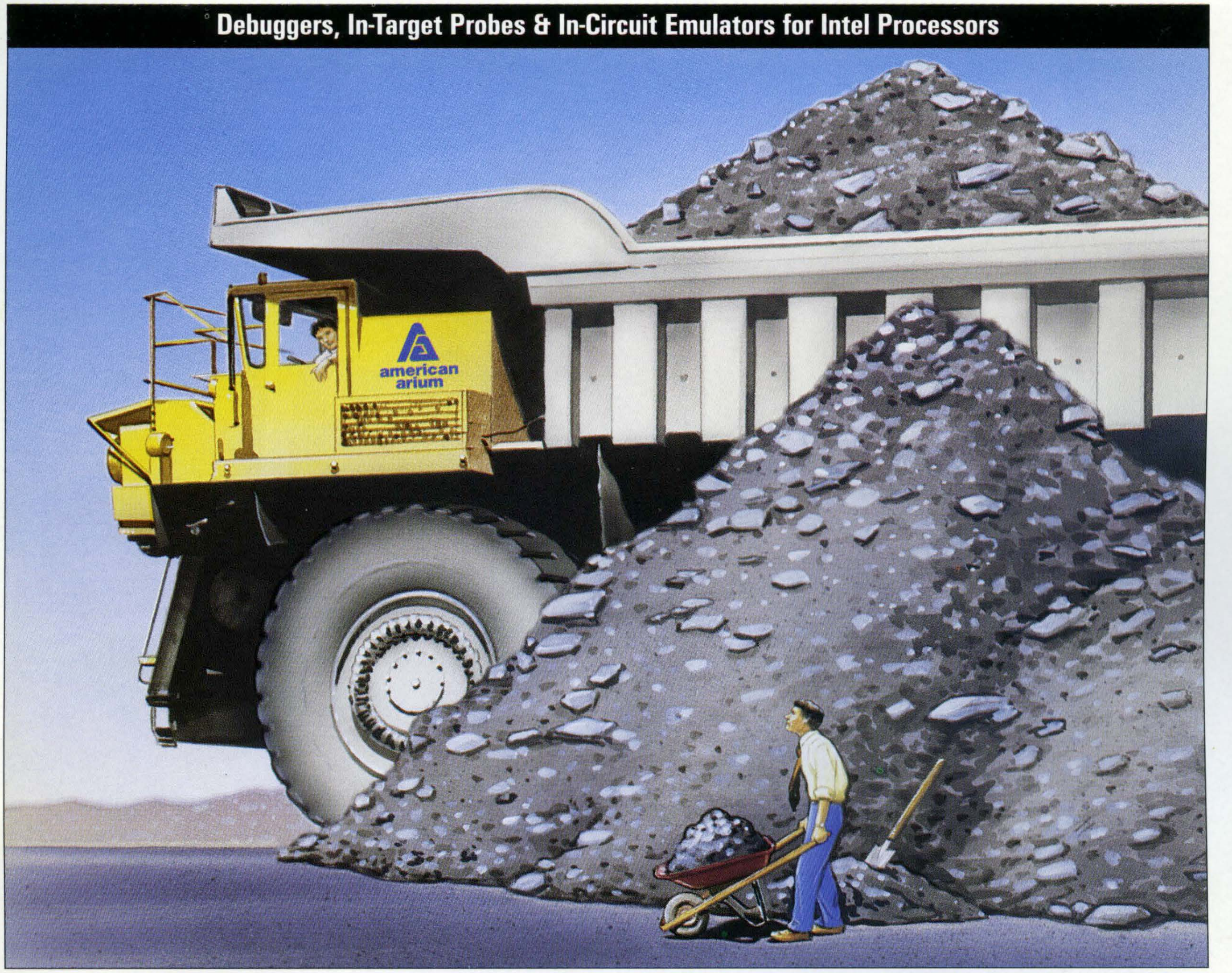
So, thank you, gentle readers, for being so gentle in your corrections to my columns, for being so involved and willing to communicate. I've tried to respond to every e-mail and will continue to do so. Don't hesitate to point out mistakes, or even better, to suggest new ideas and approaches. I hope to continue sharing this with y'all for a long time to come. **esp**

Jack G. Ganssle is a lecturer and consultant on embedded development issues. He conducts seminars on embedded systems and helps companies with their embedded challenges. He founded two companies specializing in embedded systems. Contact him at jack@ganssle.com.

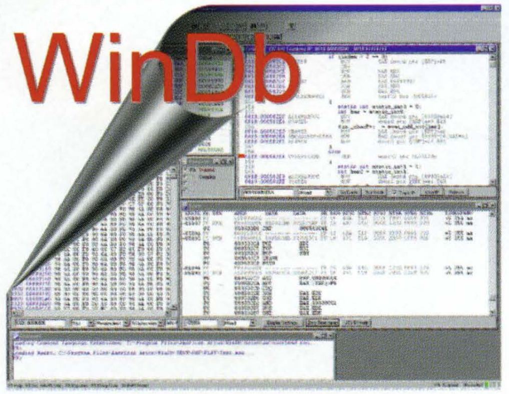
References

1. From a recently published report called "Hardware and Co-development Tools for Embedded Systems Design," by the Electronics Market Forecasters group.

Debuggers, In-Target Probes & In-Circuit Emulators for Intel Processors



It's hard to compete without the right tools.



As with most things, there's a hard way and an easy way to develop any new Intel processor based system. And just for the record, we're the guys who offer the easy way.

We've been working closely with Intel for 9 years to make sure that powerful new American Arium development tools are ready and waiting each time early silicon is released. So whether you're developing a Pentium® processor based embedded system or a server designed around the new Itanium™ processor, we've got the affordable and readily upgradeable in-circuit emulator you need. Features include:

- New! GNU support
- Unmatched breakpoints
- Industry leading debugger
- Fast setup and use
- 1 year of software upgrades and tech support

Why not call us today and put your system on the fast track to market.

american arium

14811 Myford Rd., Tustin, CA 92780
Ph: 714-731-1661 Fax: 714-731-6344
E-mail: info@arium.com

For more information on any of our in-circuit emulators, visit...

www.arium.com

Pentium and Itanium are registered trademarks of Intel Corporation.