United Business Media

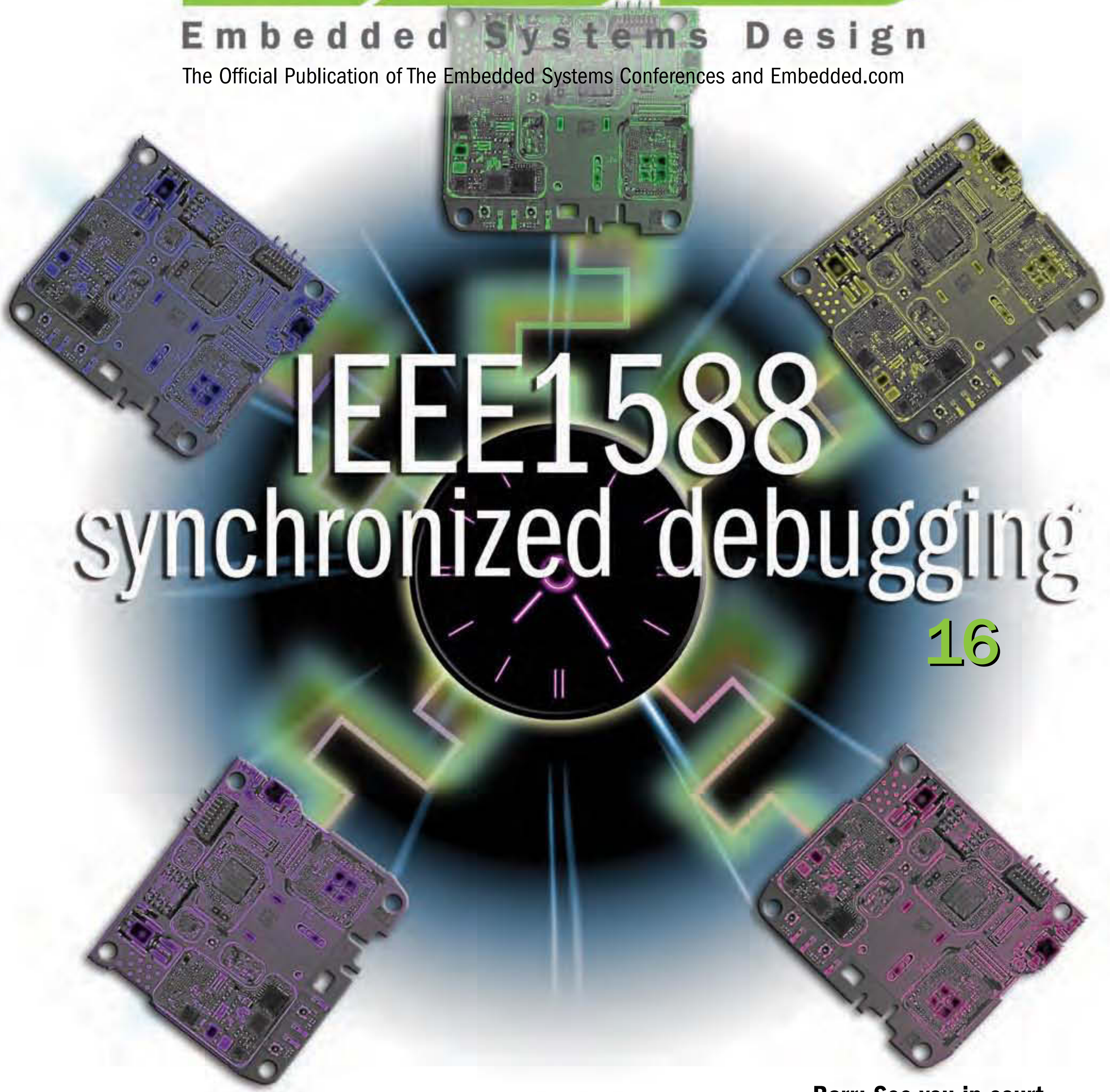
Learn today. Design tomorrow.

ESD Embedded Systems Design

The Official Publication of The Embedded Systems Conferences and Embedded.com

VOLUME 22, NUMBER 11

DECEMBER 2009



IEEE1588 synchronized debugging

16

Barr: See you in court
11

Debugging a GUI
25

Ganssle: Programming quips
38

Intel® Core™ i7

Processor-Based Mobile Platform.

6U VPX

3U VPX

6U cPCI

3U cPCI

XMC

VME

Intel® Core™ i7 Processors: Unmatched Performance

Extreme Engineering Solutions, Inc. (X-ES) unleashes the performance of the Intel Core i7 processor for use in commercial, military, and aerospace applications. The mobile Intel Core i7 processor delivers unmatched power savings and processing performance.

Extreme Engineering Solutions offers an extensive product portfolio that includes commercial and ruggedized single board computers, high-performance processor modules, multipurpose I/O modules, backplanes, enclosures, and fully integrated systems.

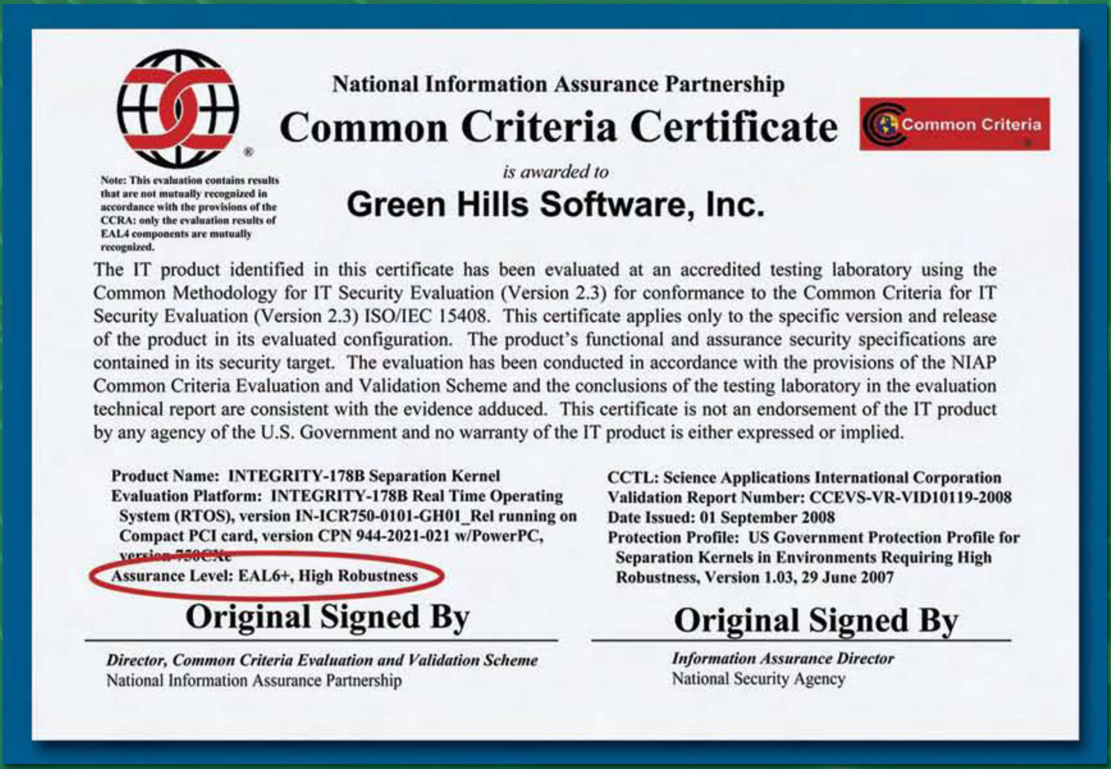
Call or visit our website today.

X-ES

Extreme Engineering Solutions

608-833-1155 • www.x-es.com

INTEGRITY RTOS has it. No one else does.



National Information Assurance Partnership

Common Criteria Certificate

Common Criteria

is awarded to

Green Hills Software, Inc.

Note: This evaluation contains results that are not mutually recognized in accordance with the provisions of the CCRA: only the evaluation results of EAL4 components are mutually recognized.

The IT product identified in this certificate has been evaluated at an accredited testing laboratory using the Common Methodology for IT Security Evaluation (Version 2.3) for conformance to the Common Criteria for IT Security Evaluation (Version 2.3) ISO/IEC 15408. This certificate applies only to the specific version and release of the product in its evaluated configuration. The product's functional and assurance security specifications are contained in its security target. The evaluation has been conducted in accordance with the provisions of the NIAP Common Criteria Evaluation and Validation Scheme and the conclusions of the testing laboratory in the evaluation technical report are consistent with the evidence adduced. This certificate is not an endorsement of the IT product by any agency of the U.S. Government and no warranty of the IT product is either expressed or implied.

Product Name: INTEGRITY-178B Separation Kernel
Evaluation Platform: INTEGRITY-178B Real Time Operating System (RTOS), version IN-ICR750-0101-GH01_Rel running on Compact PCI card, version CPN 944-2021-021 w/PowerPC, version 750CXe
Assurance Level: EAL6+, High Robustness

CCTL: Science Applications International Corporation
Validation Report Number: CCEVS-VR-VID10119-2008
Date Issued: 01 September 2008
Protection Profile: US Government Protection Profile for Separation Kernels in Environments Requiring High Robustness, Version 1.03, 29 June 2007

Original Signed By

Director, Common Criteria Evaluation and Validation Scheme
National Information Assurance Partnership

Original Signed By

Information Assurance Director
National Security Agency

The NSA has certified INTEGRITY technology to EAL6+ and High Robustness. INTEGRITY is the most secure real-time operating system available and the first and only technology to have achieved this level.

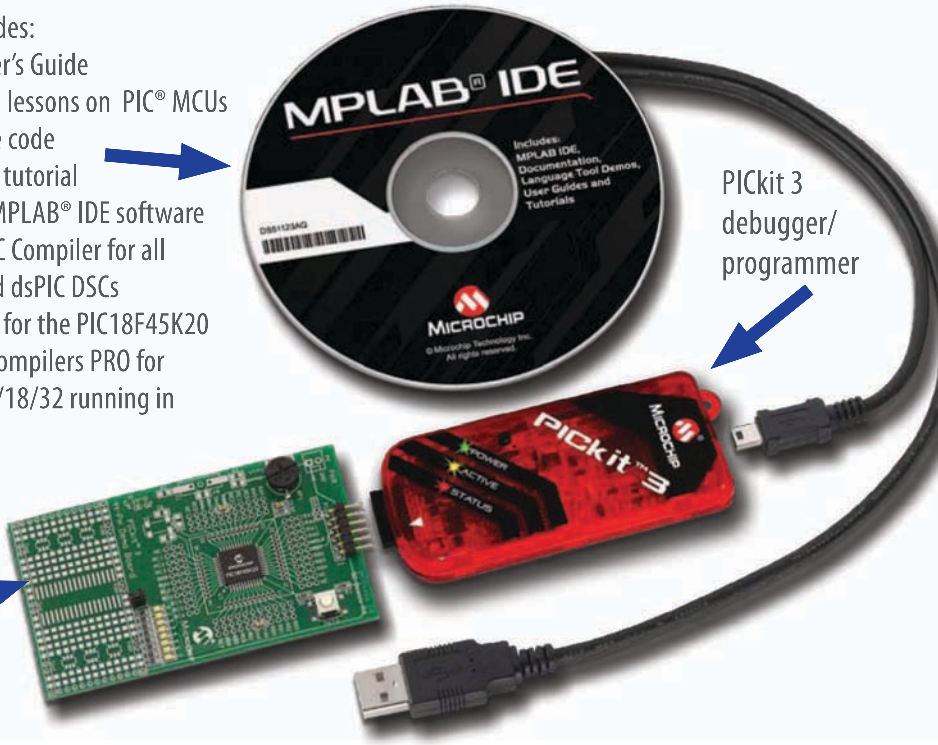
High Robustness is a higher level of security than EAL6+. It includes 133 additional security mandates—over and above the 161 required for EAL6+.

When security is required, Green Hills Software's INTEGRITY RTOS technology is the only option.

Green Hills®
• SOFTWARE, INC. •
www.ghs.com

One Debugger/ Programmer

A CD that includes:
- PICkit™ 3 User's Guide
- A series of 12 lessons on PIC® MCUs with C source code
- A debugging tutorial
- Microchip's MPLAB® IDE software
- Free MPLAB C Compiler for all PIC MCUs and dsPIC DSCs
- CCS compiler for the PIC18F45K20
- HI-TECH C® Compilers PRO for PIC10/12/16/18/32 running in Lite Mode


MPLAB® IDE
Includes: MPLAB IDE, Documentation, Language Tool Demos, User Guides and Tutorials
MICROCHIP
PICkit 3
POWER
ACTIVE
STATUS

PICkit 3 debugger/ programmer

A 44-pin demo board with a PIC18F45K20 microcontroller

One Low Price ... $69.99

Microchip's PICkit 3 Debug Express (part number DV164131) incorporates in-circuit debugging technology to provide an extremely affordable hardware debugger and programmer for the entire range of Microchip's Flash Microcontrollers (MCUs) – from the smallest 8-bit PIC10 MCU, through all 16-bit dsPIC® DSCs to the largest 32-bit PIC32 MCU.

PICkit 3 runs under the popular MPLAB IDE, the free graphical integrated development environment software, complete with programmer's editor, software simulator and project manager. MPLAB IDE's GUI promotes advanced debugging with a host of auxiliary features, such as a segmented LCD Designer, a graphics display designer and a data monitor and control interface. Easily connected to the PC with a USB interface, PICkit 3 is bundled with a demo board for fast learning and initial design prototyping. The two-wire interface easily connects to the final designs for application tuning and quick in-circuit programming.

With rich features, free compilers fully integrated with MPLAB IDE and a suite of tutorials, Microchip PICkit 3 Debug Express delivers substantial value at a remarkably low price.

For All PIC® Microcontrollers

www.microchip.com/PICkit3

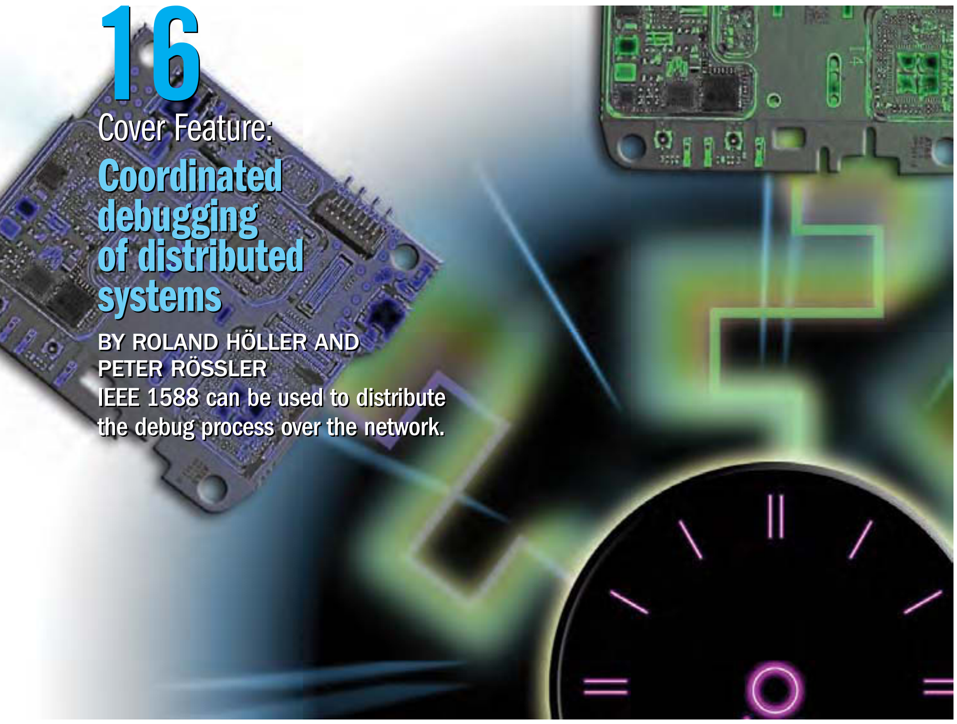
MICROCHIP

THE OFFICIAL PUBLICATION OF THE EMBEDDED SYSTEMS CONFERENCES AND EMBEDDED.COM

Learn today. Design tomorrow.

ESD EMBEDDED SYSTEMS DESIGN

VOLUME 22, NUMBER 11
DECEMBER 2009



16

Cover Feature:

Coordinated debugging of distributed systems

BY ROLAND HÖLLER AND PETER RÖSSLER

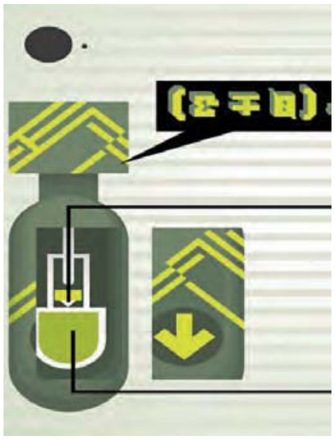
IEEE 1588 can be used to distribute the debug process over the network.

25

GUI testing—exposing visual bugs

BY NIALL MURPHY

GUI testing traditionally meant finding the most appropriate access point to inject test cases. The challenge, however, is in making the GUI tests repeatable. Here's a home-grown framework that allows test input to be managed, replacing injectable test cases.



COLUMNS

barr code 11

The lawyers are coming!

BY MICHAEL BARR

The quality of a lot of embedded software is abysmal. And lawyers are on to it. If you don't want your source code to show up in court, you better get your act together.

break points 38

Programming quotations

BY JACK G. GANSSLE

Jack Ganssle presents some of the best programming-related quips for your enjoyment and instruction.

DEPARTMENTS

#include 5

CPU vendor buys OS vendor, Part II

BY RICHARD NASS

Cavium's acquisition of MontaVista is very similar to a deal that went down not too long ago.

parity bit 7

tear down 34

marketplace 37

advertising index 37

IN PERSON

ESC Grenoble
December 1–3, 2009
www.design-reuse.com/ipesc09/

ESC Silicon Valley
April 26–29, 2010
www.embedded.com/esc/sv

ESC Chicago
June 8–9, 2010
http://esc-chicago.techinsightsevents.com/

ESC India
July 21–23, 2010
www.esc-india.com/

ESC Boston
September 20–23, 2010
www.embedded.com/esc/boston

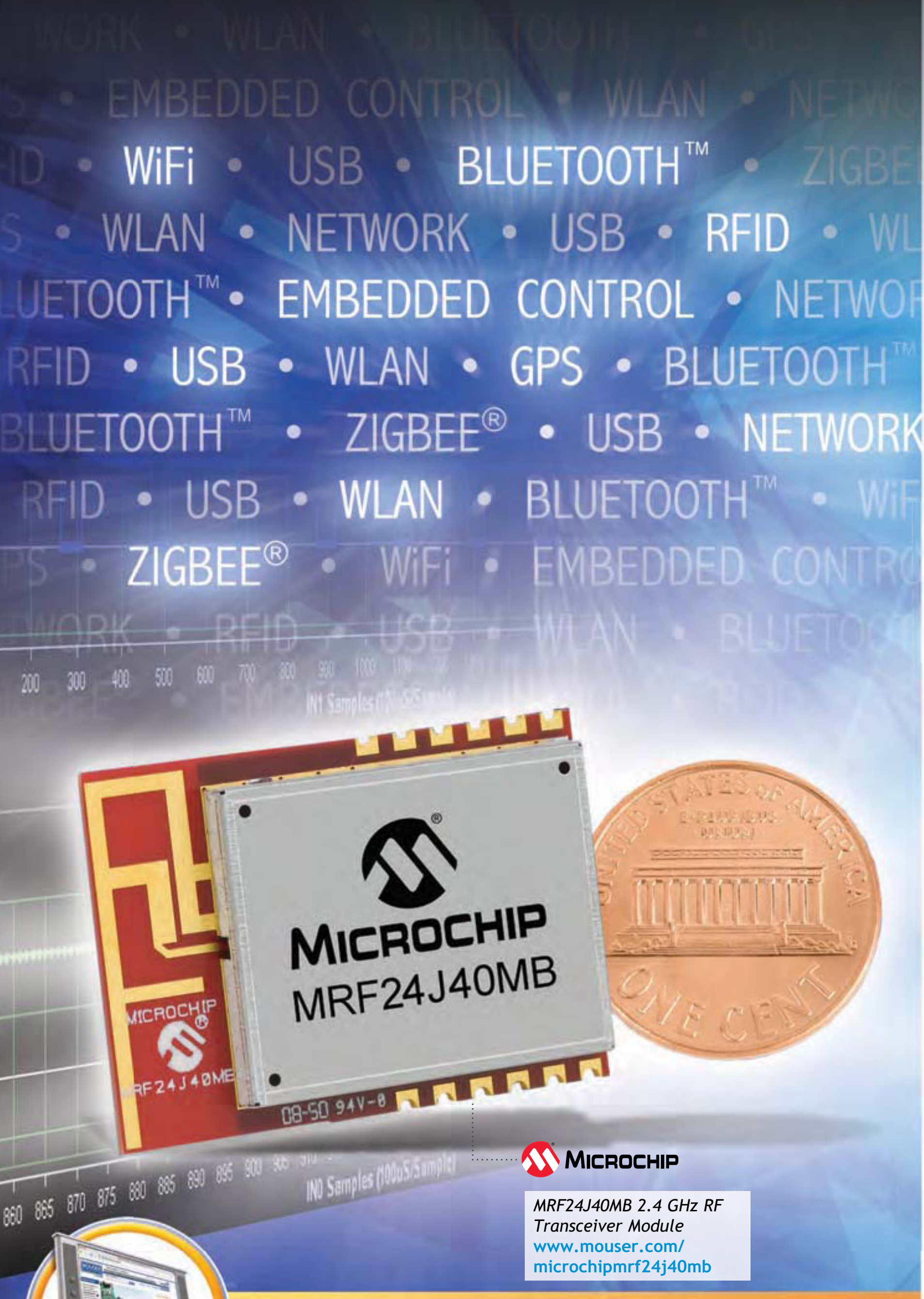
ESC UK
October 12–14, 2010
www.embedded.co.uk

ONLINE

www.embedded.com

EMBEDDED SYSTEMS DESIGN (ISSN 1558-2493 print; ISSN 1558-2507 PDF-electronic) is published monthly with the exception of a combined July/August issue by TechInsights, 600 Harrison Street, 5th floor, San Francisco, CA 94107, (415) 947-6000. Please direct advertising and editorial inquiries to this address. SUBSCRIPTION RATE for the United States is $55 for 12 issues. Canadian/Mexican orders must be accompanied by payment in U.S. funds with additional postage of $6 per year. All other foreign subscriptions must be prepaid in U.S. funds with additional postage of $15 per year for surface mail and $40 per year for airmail. POSTMASTER: Send all changes to EMBEDDED SYSTEMS DESIGN, P.O. Box 3404, Northbrook, IL 60065-9468. For customer service, telephone toll-free (877) 676-9745. Please allow four to six weeks for change of address to take effect. Periodicals postage paid at San Francisco, CA and additional mailing offices. EMBEDDED SYSTEMS DESIGN is a registered trademark owned by the parent company, TechInsights. All material published in EMBEDDED SYSTEMS DESIGN is copyright © 2009 by TechInsights. All rights reserved. Reproduction of material appearing in EMBEDDED SYSTEMS DESIGN is forbidden without permission.

The Newest Embedded Technologies


WiFi • USB • BLUETOOTH™ • ZIGBEE
WLAN • NETWORK • USB • RFID • WLAN
BLUETOOTH™ • EMBEDDED CONTROL • NETWORK
RFID • USB • WLAN • GPS • BLUETOOTH™
BLUETOOTH™ • ZIGBEE® • USB • NETWORK
RFID • USB • WLAN • BLUETOOTH™ • WiFi
ZIGBEE® • WiFi • EMBEDDED CONTROL
MICROCHIP
MRF24J40MB

MICROCHIP

MRF24J40MB 2.4 GHz RF Transceiver Module
www.mouser.com/microchipmrf24j40mb

New Products from:

RABBIT

MiniCore™ RCM5600W Wi-Fi Module
www.mouser.com/rabbit_rcm5600w

AdaptivEnergy
Joule-Thief™ Energy Harvesting

Joule-Thief™ Module
www.mouser.com/adaptivenergy_joule-thief

CEL
California Eastern Laboratories

ZICM2410P0 MeshConnect™ Module
www.mouser.com/celmesh

The ONLY New Catalog Every 90 Days

Experience Mouser's time-to-market advantage with no minimums and same-day shipping of the newest products from more than 390 leading suppliers.

MOUSER ELECTRONICS®
a tti company

The Newest Products For Your Newest Designs

www.mouser.com

Over A Million Products Online

(800) 346-6873

ESD
EMBEDDED SYSTEMS DESIGN

Director of Content/Media, TechInsights
Richard Nass
(201) 288-1904
rnass@techinsights.com

Managing Editor
Susan Rambo
srambo@techinsights.com

Contributing Editors
Michael Barr, John Canosa, Jack W. Crenshaw, Jack G. Ganssle, Dan Saks, Larry Mittag

Art Director
Debee Rommel
drommel@techinsights.com

European Correspondent
Colin Holland
colin.holland@techinsights.com

Embedded.com Site Editor
Bernard Cole
bccole@acm.org

Production Director
Donna Ambrosino
dambrosino@ubm-us.com

Subscription Customer Service
P.O. Box 2165, Skokie, IL 60076
(800) 577-5356 (toll free)
Fax: (847) 763-9606
embeddedsystemsdesign@halldata.com
www.customerserviceesp.com

Article Reprints, E-prints, and Permissions
Mike O'Brien
Wright's Reprints
(877) 652-5295 (toll free)
(281) 419-5725 ext.117
Fax: (281) 419-5712
www.wrightsreprints.com/reprints/index.cfm ?magid=2210

Publisher
David Blaza
(415) 947-6929
dblaza@techinsights.com

Editorial Review Board
Michael Barr, Jack W. Crenshaw, Jack G. Ganssle, Bill Gatliff, Nigel Jones, Niall Murphy, Dan Saks, Miro Samek

techinsights

Corporate—TechInsights

Paul Miller	*Chief Executive Officer*
Aharon Shamash	*Chief Financial Officer*
Felicia Hamerman	*Group Marketing Director*
Randall Freeborn	*Chief Human Resources Officer*
Harry Page	*Senior VP Professional Services, Semiconductor Insights, Portelligent*

United Business Media

Corporate—UBM LLC

Marie Myers	*Senior Vice President, Manufacturing*
Pat Nohilly	*Senior Vice President, Strategic Development and Business Administration*

BY Richard Nass

CPU vendor buys OS vendor, Part II

I know I'm a little late to the party on this one, but I need to put my two cents in here. When Intel made the acquisition of Wind River, I was all over it, so it's only fair that I make a comment on what looks to be a similar situation, Cavium's acquisition of MontaVista.

It's a similar situation because it's a microprocessor vendor swallowing up an operating system vendor. The only difference is that it's on a smaller scale (a significantly smaller scale). Intel reportedly paid somewhere in the neighborhood of $800 million for Wind, while Cavium paid (only) $50 million. The former was the largest processor company purchasing the largest embedded software vendor. That's not the case with Cavium-MontaVista, but the similarities are still there.

This acquisition could be good for existing MontaVista customers, as it gives them a great growth path to a very high-end microprocessor. However, it's doesn't bode well for the microprocessor vendors (except for Cavium, of course). MontaVista claims that it will continue to support all the microprocessor vendors that it currently works with, and it's a pretty long list.

Hmm. That sounds familiar. The Wind folks made that same claim. The jury is still out on Wind, but they're sticking to their story of supporting all the necessary microprocessors. Similarly, we'll have to wait and see if MontaVista holds true to their word.

Cavium and MontaVista were close partners before the acquisition, and both claim they'll work with the other's competitors. So, I ask the same question I asked a few months ago: why make this acquisition? And I come to same conclusion—over time, MontaVista will end the support for Cavium's competitors. It's a natural progression, as those competitors begin toz lose trust in MontaVista.

Think about it. If you're processor vendor X, why would you let MontaVista in on your secrets, which is required to continue support, when there's the possibility (a strong possibility?) that those secrets will find their way back to Cavium, your competition? MontaVista will claim that that won't happen, but I'm not sure if I'd be willing to take that bet if I'm vendor X and there are alternatives to MontaVista.

So, who's next? It's too late to buy Embedded Alley, as they were acquired by Mentor Graphics earlier in the year (a very smart move, in my opinion). That one was a little different, as it was an OS vendor buying another OS vendor. But there are still plenty of operating system companies available. And there are lots of processor companies who may feel the need to join the club.

Richard Nass is the director of content/media at TechInsights. You may reach him at *rnass@techinsights.com*.

Rich Nass

Richard Nass
rnass@techinsights.com

Mouser Electronics is one of the fastest growing global catalog and web-based distributors in the electronics industry, Mouser is dedicated to supplying design engineers with the most rapid introduction of the newest products, leading-edge technologies, and world class customer service. Focused on design engineers and buyers demanding small to medium quantities of the latest products, Mouser provides customer-focused distribution.

Rapid New Product Introduction

Mouser delivers a time-to-market advantage to customers with the industry's only NEW 2,100+ page print catalog published every 90 days, featuring the newest products and technologies. Continuous updates are made to ensure the newest products are added and end-of-life products are removed from the catalog, providing customers with easy access to the newest products and technologies.

E-Commerce

Mouser.com is updated daily with new products and technologies available for customers to browse and buy online. The company's website features **OVER A MILLION SEARCHABLE PRODUCTS,** more than 900,000 downloadable data sheets, 423+ supplier catalogs, and 1.5+ million cross-referenced parts. The constant refinement to the site includes technical design information, as well as numerous user-friendly tools such as Project Manager with automatic reorder, BOM import capabilities, automatic order confirmation, and live chat in several languages.

Broad Product Line

Mouser is the design engineer's one-stop shop for all the board-level components and associated development tools necessary for total project design. The company's broad-based linecard consists of components across the board from more than 390 industry-leading manufacturers of semiconductors, optoelectronics, embedded modules, as well as passives, interconnects, electromechanical, circuit protection devices, enclosures, thermal management, and wire/cable products.

State-of-the-Art Warehouse Operations

Mouser's global corporate headquarters facility is based in Texas, USA, and totals 432,000 sq. ft. encompassing offices, call and data centers, as well as its state-of-the-art warehouse. The wireless warehouse management system is streamlined to nearly perfect pick-and-ship operations. These high-tech capabilities and the efficient order fulfillment processes allow for fast delivery with same-day shipping on most orders received by 8 p.m. CST.

The 3,400 sq. ft. data center features a pre-action fire suppression system, connections to multiple power sources, and a dedicated fiber communications ring to ensure uninterrupted internet operations – a core component of Mouser's global sales operations.

Worldwide Customer Service

Mouser is dedicated to providing superior service and support to customers worldwide. In addition to the United States locations, Mouser has sales offices around the world, including Singapore, Shanghai, and Hong Kong, as well as Germany, UK, and Israel.

Mouser believes in providing the best customer service– regardless of the size of the customer or the size of the order. Breaking packs and no minimum orders, including one-piece shipping, is especially attractive to engineers working in the earliest stages of the engineering design cycle.

MOUSER® ELECTRONICS

a tti company

The Newest Products For Your Newest Designs

www.mouser.com (800) 346-6873

CAN it be possible

While it is nice to see an article on CAN, I'm not sure whether it really lives up to its title. *(Hassane El-Khoury, "CAN in 30 minutes or less," November 2009, p. 20, www.embedded.com/220900314)* Some CAN history, bit-timing and tool screenshots, but not a single example of where such tools are found or how to setup CAN from scratch with them.

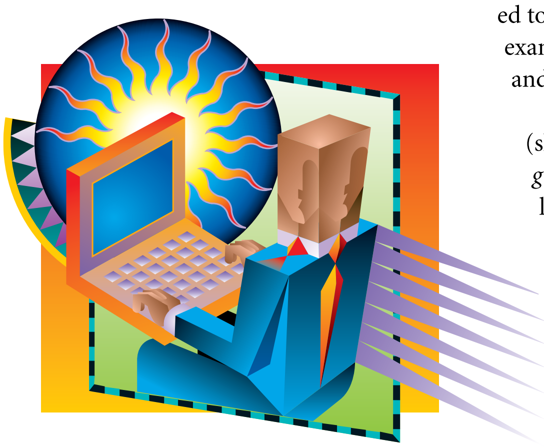
Nonetheless, the author is absolutely right when it comes to setting up CAN for the first time: while the protocol is robust, flexible, and with the Full/Enhanced controllers takes hardly any effort for your MCU, the initial design can be very tricky. There are some free implementations available (Pelican, VSCP, CAN-Open, DeviceNet), but when you want to do something simple (where these protocols simply don't fit), where a bus network like CAN seems appropriate instead of Peer to peer RS232 or maybe bus RS 485/422, the very advantage CAN has in the form of message-based communications, becomes a burden while designing your app around it. Especially, as the author also points out, when it comes to combining the various incarnations of CAN (Basic, Full, enhanced).

—Dirk Buijsman
Lead Software Engineer

The article was good, but a small erratum SPI—mentioned as "System Packet Interface" is a network protocol. I guess the author intends to print "Serial Peripheral Interface."

—Raj Miriyala
Firmware Engineer

Thank you very much for this nice article. I would like to bring up a little point that J1939 implementers will appreciate. According to SAE spec, J1939 has to run at 250 kbit/sec. This makes J1939 bit timing to 4 microseconds. Since 1 bit is 8 to 25 time quanta, make sure your oscillator can run at a perfect frequency so that your CAN device can meet the SAE spec.



When it comes to setting up CAN for the first time: while the protocol is robust, flexible, and with the Full/Enhanced controllers takes hardly any effort for your MCU, the initial design can be very tricky.

—Umut Tezduyar
Software Engineer

Doctors and debugging

I was quite interested in Jack Ganssle's use of the medical diagnosis analogy *(Jack Ganssle, "Developing a good bedside manner," October 2009, p. 38, www.embedded.com/220100899)*. I wrote the book *Debugging*, published in 2002 and still selling well because it extracts the essence of debugging, which as you point out, is not restricted to hardware and software. I use examples from medicine, car repair, and plumbing, to name a few.

I came up with nine rules (shown on the website *debuggingrules.com* in a free poster). I challenge anyone to prove that I've included a rule you can ignore or that I've omitted a rule. Your six steps (and other important things) are covered by my nine rules, except for hypothesis—fix—test sequence, with which I respectfully disagree.

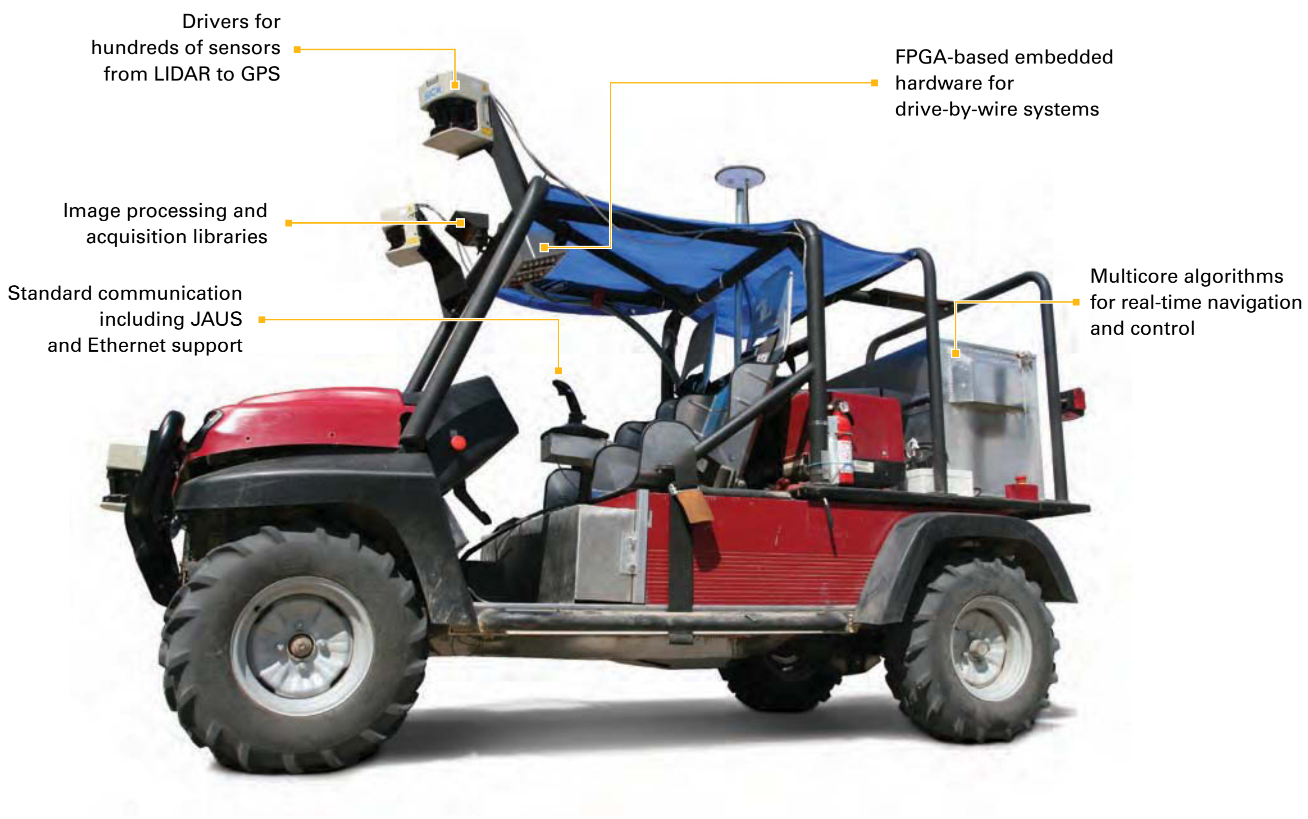
My rule #3: "Quit thinking and look" means use your hypothesis to decide where to look next, not what fix to try. Trying a fix before you have *seen* the cause of the bug is sometimes effective, but often leads to a long loop of misdirected fixes. (There are examples in the book.) The other rules are equally important, in fact, here they are:

- Understand the system
- Make it fail
- Quit thinking and look
- Divide and conquer
- Change one thing at a time
- Keep an audit trail
- Check the plug
- Get a fresh view
- If you didn't fix it, it ain't fixed

—Dave Agans
Engineer

NI LabVIEW

Limited Only by Your Imagination



Drivers for hundreds of sensors from LIDAR to GPS
Image processing and acquisition libraries
Standard communication including JAUS and Ethernet support
FPGA-based embedded hardware for drive-by-wire systems
Multicore algorithms for real-time navigation and control

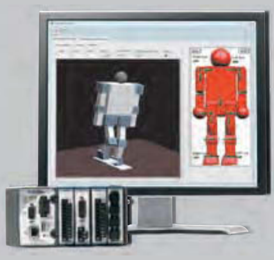
RF

Medical

Robotics

Multicore

LabVIEW graphical programming software and modular NI hardware, such as CompactRIO and PXI, are helping engineers develop fully autonomous robotics systems, including unmanned vehicles designed to compete in DARPA Grand Challenge events.



PRODUCT PLATFORM

NI LabVIEW graphical and textual programming

NI CompactRIO embedded control hardware

NI LabVIEW Real-Time Module

NI LabVIEW FPGA Module

>> Find out what else LabVIEW can do at **ni.com/imagine/robotics** **866 337 5041**

NATIONAL INSTRUMENTS™

Jack Ganssle responds: Also do check out Steve Litt's site, troubleshooters.com.

In Jack's column "*Developing a good bedside manner,*" he writes: "In other cases, just as in medicine, one bug may present a variety of odd effects. Or a single symptom could stem from a combination of bugs all interacting in excruciating-complex, and hard to diagnose, manners. I wonder if physicians observe the infrequent symptoms we see, that appear in a system once, go away for weeks, and then randomly resurface?"

I liked being the Hero that caught the elusive bug as much as anyone. But do we really have to let the bugs "go away for weeks"? How much time and money are spent chasing these bugs? How much does it cost when we fail to catch them? Are we not smart people, with systems of our own design and under our own control?

These bugs can be easily captured, if we make proper use of our software to help us. The vast majority of embedded systems can be "instrument" (in software by the developer) to record and then replay the software execution. The data rate of a proper implementation is surprising low (~2KB per MHz of CPU clock). A rate that is lower than typical instrumentation approaches that pump out information that we think will help us find these bugs.

The record process saves the minimum data that is needed to capture the exact execution process of the software. Therefore the real-time execution is not being changed by the analysis and debug processes.

The replay process recreates the recorded execution with the bugs. Complete analysis and debugging takes place in the replay process without changing the recreated execution of the software.

So what's the big disadvantage?

It requires a change in the typical embedded mindset!

—*Robert Coker*
Former Embedded Systems Engineer

ARM wrestling

Rich Nass says "it's likely that there will be no clear cut winner" *(Richard Nass, "15 billion sockets up for grabs," November 2009, p.5, www.embedded.com/220900316).* It's clear to me . . . ARM WINS.

—*Leandro Gentili*
Software Engineer

In terms of safety, gazing at the code is analogous to organizing the deckchairs on the Titanic as you steam toward the Requirements Iceberg—at least it keeps you occupied.

(ARM == less performance) ??? I do not think so. The ARM architecture appears to be dominating the small device space by a pretty decent margin too!

— *Ken Wada*
Sr. Embedded Systems Consultant

DO-178B

To set the record straight on DO-178B: Aviation incidents caused by software systems are thankfully rare, even given the exponential growth of software density in aircraft systems—perhaps this is a consequence of prescriptive standards (guidelines) such as DO-178B, and their enforcement by certification authorities (e.g., FAA Designated Engineering Representatives). A major criticism here *(Jack Ganssle, "Software for dependable systems," November 2009, p. 37, www.embedded.com/220900315)* of the book *Software for Dependable Systems* is it is not prescriptive enough!

DO-178B is far from perfect, but it has known strengths, such as its bias toward requirements and their satisfaction. The guideline contains more than 60 objectives. Many people concentrate on a select number of code-based objectives (e.g. MC/DC coverage), but this is a gross misinterpretation and misrepresentation of DO-178B.

DO-178B is limited in scope, and it fails to address the holistic system aspects of safety. Still, this article falls into a similar trap, with too much emphasis latterly on coding languages (such as Ada and SPARK). Note: systems containing perfectly functioning code have contributed to fatal accidents (such as Cali, Colombia).

In terms of safety, gazing at the code is analogous to organizing the deckchairs on the Titanic as you steam toward the Requirements Iceberg—at least it keeps you occupied.

The most pertinent statement in the article is, "Finally, expertise is demanded." Competence and professionalism should not be implicit or assumed when the systems are high-dependability. Interestingly, standard IEC 61508 provides guidance on competence assessment.

For interested readers, there is a DO-178B group on LinkedIn.

—*Martin Allen*
Software Safety Specialist

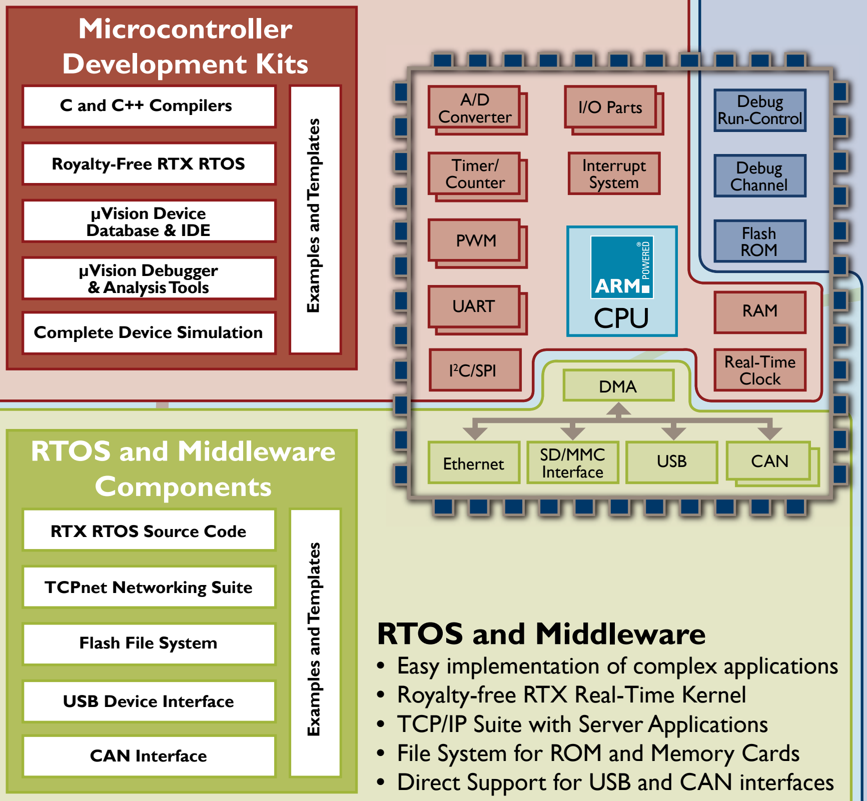
We welcome your feedback. Letters to the editor may be edited. Send your comments to Richard Nass at rnass@techinsights.com or fill out one of our feedback forms online, under the article you wish to discuss.

Development Solutions

KEIL™ Tools by ARM

for ARM, 8051 & XE166 Microcontrollers

C and C++ Development Tools
- Best-in-class Keil and ARM compilers for small, fast code
- Genuine Keil µVision® IDE/Debugger/Simulator
- Fast development and verification using Device Simulation
- Easy device configuration with Device Database support for more than 1700 devices



ULINK® USB Adapters
- JTAG, SWD & SWV support
- Debugging and Flash download
- ARM device support
- Data Trace for Cortex-M3
- ARM Instruction Trace (ULINK*Pro*)
- 8 and 16-bit support (ULINK2)

Evaluation Boards

Keil provides a wide range of evaluation boards for 8, 16 and 32-bit devices

RTOS and Middleware
- Easy implementation of complex applications
- Royalty-free RTX Real-Time Kernel
- TCP/IP Suite with Server Applications
- File System for ROM and Memory Cards
- Direct Support for USB and CAN interfaces

Keil PK51, PK166, and MDK-ARM
support more than 1,700 microcontrollers
keil.com/dd

Download the µVision4 Beta Version
keil.com/uv4

Keil RL-ARM and ARTX-166
highly optimised, royalty-free middleware suites
keil.com/rtos

keil.com
1-800-348-8051

By Michael Barr

The lawyers are coming!

When I started writing firmware and for years afterward, few people outside of the electronics design community gave a thought to the countless embedded systems around them. At the time, I found it difficult to explain to most friends and relatives what exactly it was that I did for a living. Yet embedded software was all around them at home and at work—in their phones, anti-lock brakes, laser printers, and many other important products. But to these folks, "software" was something you bought in a box at a store and installed on the one "computer" you owned.

Today, of course, there are countless embedded systems per person, and our health and wellbeing are both greatly enriched by and increasingly dependent upon proper functioning of the firmware inside. Consumers now notice them and think of them as software containers—if only because they require frequent reboots and upgrades. And there is no let up in sight: several billion more such devices are produced each year.[1]

Lawsuits are on the rise, too. In recent years, I've been called into U.S. District Court (as an expert witness) in several dozen lawsuits involving embedded software. I've met others with similar experiences and become aware of many other cases. Popular claims range from copyright theft and patent and trade secret infringement to traditional products liability with a firmware twist. Unfortunately, the quality and reliability of our collective firmware leaves the door open to an ever-increasing number of the latter.

The quality of a lot of embedded software is abysmal. And lawyers are on to it. If you don't want your source code to show up in court, you better get your act together.

THIS CODE STINKS!

At a recent Embedded Systems Conference, I gave a popular free talk titled "This Code Stinks! The Worst Embedded Code Ever" in which I used lousy code from real products as a teaching tool. The example code was gathered by a number of engineers from a broad swath of companies over several years.[2]

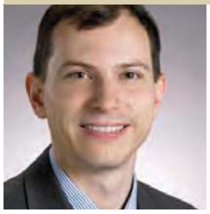
Listing 1 shows just one example of the bad code in that presentation. I don't know if the snippet contains any bugs, as most of the other examples were found to. And that's a problem. Where are we supposed to begin an analysis of the code in Listing 1? What is this code supposed to do when it works? What range of input values is appropriate to test? What are the correct output values for a given input? Is this code responsible for handling out-of-range inputs gracefully?

The original listing had no comments on or around this line to help. I eventually learned that this code computes the year, with accounting for extra days in leap years, given the number of days since a known reference date (such as January 1, 1970). But I note that we still don't know if it works in all cases, despite it being present in an FDA-regulated medical device. I note too that the Microsoft Zune Bug[3] was buried in a much better formatted snippet of code that performed a very similar calculation.

Listing 2 contains another example, this time in C++, with the bug-finding left as an exercise for the reader. You can find the full set of slides from my talk online at *http://bit.ly/badcode*.

TOTAL RECALL

Lest you think that the evidence from the presentation are



Michael Barr is the author of three books and over 50 articles about embedded systems design, as well as a former editor in chief of this magazine. Michael is also a popular speaker at the Embedded Systems Conference and the founder of embedded systems consultancy Netrino. You may reach him at mbarr@netrino.com or read more by him at www.embeddedgurus.net/barr-code.

Listing 1 A single line of unintelligible mystery code.

```
y = (x + 305) / 146097 * 400 + (x + 305) % 146097 / 36524 * 100 + (x + 305) % 146097 % 36524
    / 1461 * 4 + (x + 305) % 146097 % 36524 % 1461 / 365;
```

exceptions to the norm found because I and other engineers were on the prowl for bad code, consider just a couple of examples stemming from the more obvious embedded software failures.

First, recall the Patriot Missile failure in Dhahran, Saudi Arabia during the first Gulf War. Twenty-eight U.S. soldiers were killed when a Scud missile was not shot down due to improper tracking by the Patriot Missile battery protecting a military base. A report from the U.S. Government Accountability Office examined the events leading to the failure and concluded the problem was partly in the requirements: the government didn't tell the designer it would need to "operate continuously for long periods of time." Huh!? "At the time of the incident, the battery had been operating continuously for over 100 hours".[5, 6]

Now consider a more recent example. GPS-maker Garmin announced a "free, mandatory GPS software update to correct a software issue that has been discovered to cause select GPS devices to repeatedly attempt to update GPS firmware and then either shut down or no longer acquire GPS satellite signals." This sounds to me like a bug in their bootstrap loader (a.k.a., bootloader). Many Garmin GPS units are named as affected, including members of the popular nüvi product family.[7]

Or consider what a consumer had to say about his Celestron SkyScout Personal Planetarium recently in a forum at Amazon.com: "I'm downloading the second firmware update release since I've had my SkyScout . . . about 3 weeks. Each release is making the device more stable."

Listing 2 Find the bug in this code snippet.[4]

```
bool Probe::getParam(uint32_t param_id, int32_t idx)
{
  int32_t  val  = 0;
  int32_t  ret  = 0;

  ret = m_pParam->readParam(param_id, idx, &val);

  if (!ret)
  {
    logMsg("attempt to read parameter failed\n");
    exit(1);
  }
  else …
```

Finally, consider these quotes from the recent recall of a device regulated by the U.S. Food and Drug Administration—an AED (automatic external defibrillator):

- "Units serviced in 2007 and upgraded with software version 02.06.00 have a remote possibility of shut down during use in cold environmental conditions. There are no known injuries or deaths associated with this issue. The units will be updated with the current version of software."
- "All of the recalled units will be upgraded with software that corrects [another] unexpected shutdown problem. In the meantime . . . it is vital to follow the step 1-2-3 operating procedure which directs attachment of the pads after the device has been turned on. This procedure is described on the back of your device and also in the Quick Reference material inside the AED 10 case. Some pages in the user's manual may erroneously describe or show illustrations of [a different] operating procedure . . . Please disregard these erroneous instructions."

At least one death was reported at a time when the second type of unexpected software shutdown occurred. Are bugs in the embedded software to blame for that too? If not, how did the User's Manual come to be out of sync with the firmware in a process-driven FDA-regulated environment?

Given the above, is it *not* appropriate to wonder if the unexplained loss of Air France 447 over the Atlantic Ocean earlier this year was firmware-related? An abrupt 650-ft. dive an Airbus A330 flight experienced in October 2006 may offer clues to the loss of Air France 447. Authorities have blamed a pair of simultaneous computer failures for that event in the fly-by-wire A330. First, one of three redundant air data inertial reference units began giving bad data. Then, a voting algorithm intended to handle precisely such a failure in one unit by relying only on the other two failed to work as designed; the flight computer instead made decisions only on the basis of the one failed unit! "More than 100 of the 300 people on board were hurt, with broken bones, neck and spinal injuries, and severe lacerations splattering blood throughout the cabin."[8] A lawsuit is pending.

TAKE A DEEP BREATH

Firmware bugs seem to be everywhere these days. So much so that firmware source-code analysis is even

When it comes to creating your next embedded system, an important decision awaits you. Memory. Your design needs speed, reliability, performance and capacity to store the code and data your design demands. No problem. Numonyx has the broadest portfolio of parallel and serial NOR, NAND and phase change memory. And we offer extended temperature support with AEC-Q100 certification on many of our products and expanded design versatility with voltage support up to 5V. All designed to deliver a right-fit solution to help you shorten design cycles, reduce development costs and accelerate the roll-out of your next big idea. Find out how Numonyx memory matters for your next design.

NUMONYX® FORTÉ™ SERIAL FLASH MEMORY

Product Family	Density Range	Voltage/Solution
M25P (block erase)	512 k - 128 Mb	3V, single-I/O
M25PX (4KB block erase)	4 Mb - 64 Mb	3V, multi-I/O
M25PE/M45PE (page erase)	1 Mb - 16 Mb	3V

NUMONYX® AXCELL™ PARALLEL NOR FLASH MEMORY

Product Family	Density Range	Voltage/Solution
M29W/EW (JEDEC command set)	4 Mb - 2 Gb*	3V, page read†
P30/33 (Intel-based command set, sync burst)‡	64 Mb - 2 Gb*	1.8/3V core

Visit www.Numonyx.com/Embedded
for free access to the Numonyx Embedded Design Center.

numonyx™
innovative. memory. solutions.

*Includes stacked solutions †Available on specific densities ‡Other names and brands may be claimed as the property of others entering the courtroom in criminal cases involving data collection devices with software inside. Consider the precedent-setting case of the Alcotest 7110. After a two-year legal fight, seven defendants in New Jersey drunk driving cases successfully won the right to have their experts review the source code for the Alcotest firmware.[9]

The state and the defendants both ultimately produced expert reports evaluating the quality of the firmware source code. Although each side's experts reached divergent opinions as to the overall code quality, several facts seem to have emerged as a result of the analysis:

- Of the available 12 bits of analog-to-digital converter precision, just 4 bits (most-significant) are used in the actual calculation. This sorts each raw blood-alcohol reading into one of 16 buckets. (I wonder how they biased the rounding on that.)
- Out of range A/D readings are forced to the high or low limit. This must happen with at least 32 consecutive readings before any flags are raised.
- There is no feedback mechanism for the software to ensure that actuated devices, such as an air pump and infrared sensor, are actually on or off when they are supposed to be.
- The software first averages the initial two readings. It then averages the third reading with that average. Then the fourth reading is averaged in, and so on. No comments or documentation explains the use of this formula, which causes the final reading to have a weight of 0.5 in the "average" and the one before that to have a weight of 0.25, and so forth.
- Out of range averages are forced to the high or low limit, too.
- Static analysis with lint produced over 19,000 warnings about the code (that's about three errors for every five lines of source code).

What would you infer about the reliability of a defendant's blood-alcohol reading if you were on that jury? If you're so inclined, you can read the full expert reports for yourself.[10]

A BETTER WAY

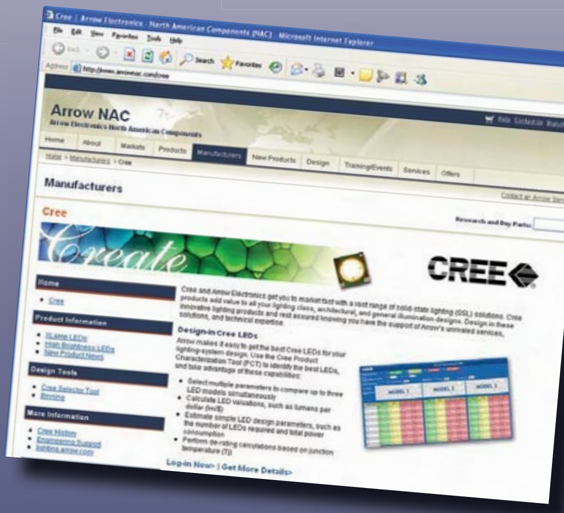
Don't let your firmware source code end up in court! Adopt a coding standard that will prevent bugs and start following it; don't wait a day. Run lint and other static analysis and code complexity tools yourself, rather than waiting for an expert witness to do it for you. Make peer code reviews a regular part of every working day on your team. And establish a testing environment and regimen that allows for regression testing at the unit and system level. These best practices won't ensure perfect quality, but they will show you tried your best.

I'll have more to say about keeping bugs out of embedded software in my next few columns. Meanwhile, try not to think about all the firmware upon which your life depends. ■

ENDNOTES:

1. *www.vdcresearch.com/_documents/press-release/press-attachment-1503.pdf*
2. Minor details, including variable names and function names, were changed as needed to hide the specifics of applications, companies, or programmers.
3. *http://bit-player.org/2009/the-zune-bug*
4. Hint: This code was embedded in a piece of factory automation equipment.
5. GAO's report can be found at *www.fas.org/spp/starwars/gao/im92026.htm*.
6. In fact, soldiers in Israel had previously discovered that the Patriot Missile software's ability to track an incoming missile degraded in just eight hours, and they had a software upgrade to fix it.
7. *http://garmin.blogs.com/my_weblog/2009/06/ask-garmin-free-mandatory-gps-software-available-now.html*
8. *www.time.com/time/world/article/0,8599,1902421,00.html*
9. *www.dwi.com/new-jersey/state-v-chun*
10. Full expert findings reports: *www.dwi.com/new-jersey/base-one-findings* (defendants) and *www.dwi.com/new-jersey/new-jersey/code-review* (state).

Create with Arrow and Cree



Cree | Arrow Electronics - North American Components (NAC) - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address http://www.arrownac.com/cree

Arrow NAC
Arrow Electronics North American Components

Home | About | Markets | Products | Manufacturers | New Products | Design | Training/Events | Services | Offers

Home > Manufacturers > Cree

Manufacturers

Cree

Research and Buy Parts:

Home
- Cree

Product Information
- XLamp LEDs
- High Brightness LEDs
- New Product News

Design Tools
- Cree Selector Tool
- Binning

More Information
- Cree History
- Engineering Support
- lighting.arrow.com

Cree and Arrow Electronics get you to market fast with a vast range of solid-state lighting (SSL) solutions. Cree products add value to all your lighting class, architectural, and general illumination designs. Design in these innovative lighting products and rest assured knowing you have the support of Arrow's unrivaled services, solutions, and technical expertise.

Design-in Cree LEDs

Arrow makes it easy to get the best Cree LEDs for your lighting-system design. Use the Cree Product Characterization Tool (PCT) to identify the best LEDs, and take advantage of these capabilities:

- Select multiple parameters to compare up to three LED models simultaneously
- Calculate LED valuations, such as lumens per dollar (lm/$)
- Estimate simple LED design parameters, such as the number of LEDs required and total power consumption
- Perform de-rating calculations based on junction temperature (Tj)

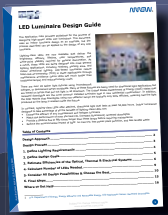
Log-in Now> | Get More Details>

Everything You Need to Design–in Cree LEDs

Arrow and Cree have built an intuitive, information-packed website to help you design the optimal Cree LEDs into your solid-state lighting applications. Get the facts you need to know, and then select the right Cree LEDs for your lighting-class, architectural, and general illumination designs. With Arrow's world-class support, you can rest assured you're choosing the best products and benefiting from unrivaled services, solutions, and technical know-how.

Visit our site, and learn all about:

- New Cree LEDs
- Solutions that will get you to market fast
- Relevant white papers and other technical papers
- Training opportunities
- Related design tools



LED Luminaire Design Guide

Download the Cree Lighting Design Guide!

Go to **www.arrownac.com/cree** to learn about designing high-power LEDs into luminaires.

Our site also allows you to access Cree's Product Characterization Tool to:

- Compare up to three LED models simultaneously
- Calculate lumens per dollar and other valuations
- Identify the best Cree LEDs for your design

Visit www.arrownac.com/cree to get started now.

CREE®

1.888.9LIGHT1 | www.arrownac.com/cree

ARROW®

IEEE 1588 can be used to distribute the debug process over the network.

Coordinated debugging of distributed systems

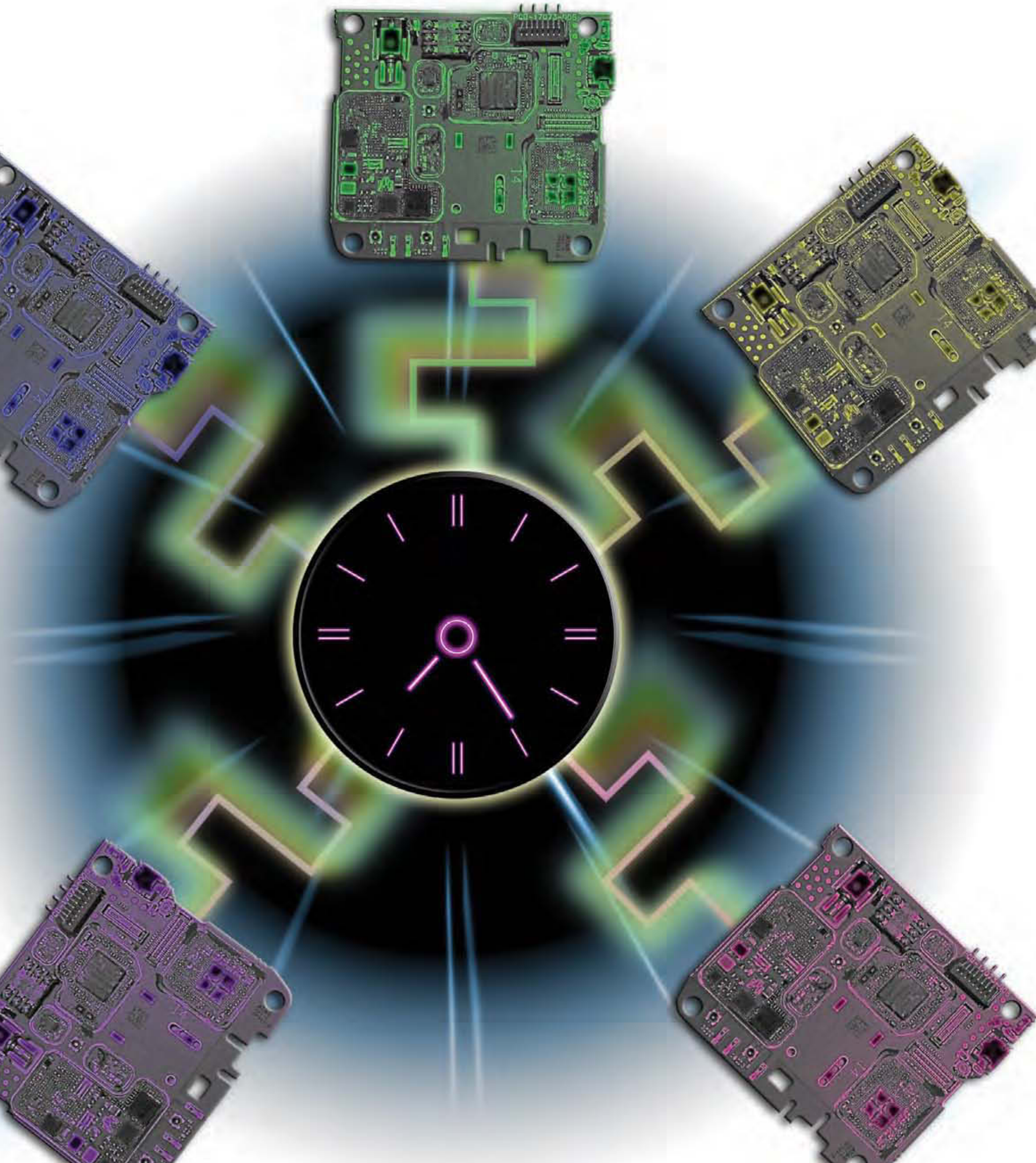
BY ROLAND HÖLLER AND PETER RÖSSLER

Imagine a world without a global notion of time. Now try to find out the flight direction of an airplane with the following information: There's an e-mail from Alice that she saw the plane about two hours after sunrise and another e-mail from Bob that he saw the plane about three hours after sunrise. So Alice and Bob tell us when they saw the plane, at least from their point of view. If they are nice, they might give us some additional information, namely their location at the moment of the observation. But, unfortunately embedded systems are usually not that nice.

Now imagine a distributed system built of networked embedded nodes. When a problem arises with the distributed application, the designer invokes a debugger to find out the faulty system behavior. In detail, the designer traces the execution of two nodes A and B simultaneously. The situation is similar to the plane-tracking scenario. Obviously, a systemwide notion of time would be helpful, which leads us to an important aspect in distributed debugging.

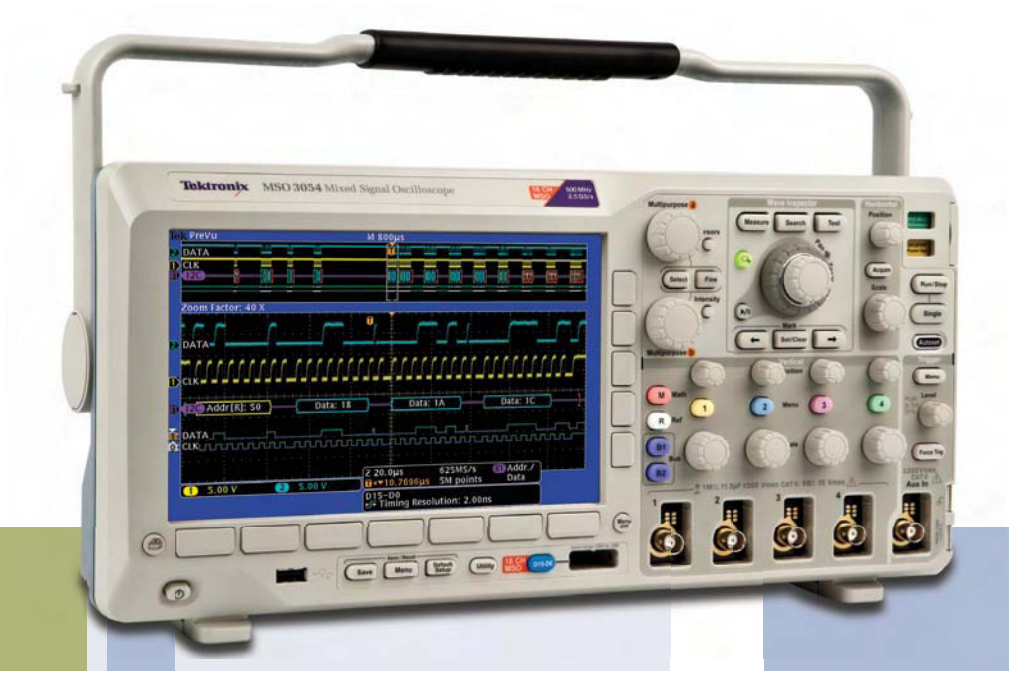
STATE-OF-THE-ART EMBEDDED SYSTEMS DEBUGGING

To alleviate the difficulty of debugging of modern microcontrollers and complex system on chips (SoCs), support for test and debugging is routinely



It’s More Than Just a Scope

The all-in-one solution for mixed signal debug.



Analyze analog and digital signals with the NEW MSO3000 Series.

Now you can have it all with the Mixed Signal Oscilloscope Series from Tektronix. With as many as 20 channels for analyzing analog and digital signals, you can simultaneously monitor many points of your design. And with automated decode for both parallel and serial buses, you can instantly see what all those bits mean. Use Wave Inspector® to speed through your entire waveform in seconds or to automatically search for an event you specify, even serial packet content. It’s the complete, all-in-one solution to debug today’s complex designs – fast. And with the addition of the NEW MSO3000 Series, the family just got better. See for yourself. View the virtual product demo.

Mixed Signal Oscilloscope Series

Features	4000 Series	New! 3000 Series	2000 Series
Bandwidth	1 GHz, 500 MHz, 350 MHz	500 MHz, 300 MHz, 100 MHz	200 MHz, 100 MHz
Analog Channels	4	2 or 4	2 or 4
Digital Channels	16 (MSO Series)	16 (MSO Series)	16 (MSO Series)
Record Length	10 M points	5 M points	1 M points
Display	10.4"	9"	7"
Serial and Parallel Bus Analysis	I²C, SPI, CAN, LIN, FlexRay, RS-232, Audio, Parallel	I²C, SPI, CAN, LIN, RS-232, Audio, Parallel	I²C, SPI, CAN, LIN, RS-232, Parallel
Optional Analysis Packages	Power Analysis, HDTV and Custom Video	Power Analysis, HDTV and Custom Video	–

See the family in action. View the virtual product demo: www.tektronix.com/thesolution

Tektronix®

built into silicon. Today, many debugging approaches rely on offline debugging based on trace buffers added to the CPU to reduce intrusiveness by the debug system. Leading processor-core vendors offer on-chip trace solutions.

However, existing debugging and test tools are mainly focused toward single nodes with one or more CPUs on-board or on-chip (SoC) by using auxiliary debug interfaces like JTAG or a simple UART. The problem of these approaches is that they entirely neglect the distributed nature of many applications since to connect a monitoring computer directly to each node is impractical, especially if the nodes are already embedded in their place of installation (see **Figure 1a**).

Wouldn't it be nice to precisely coordinate debug, test, trace, and replay activities across the entire distributed system without the use of any auxiliary interface or special cabling? Moreover, it would be helpful if only a single debugging master and monitoring computer is attached to the network, used to issue debugging, test, or monitoring actions. Such an approach that greatly simplifies debugging and testing of distributed systems is shown in **Figure 1b**.

Wouldn't it be nice to precisely coordinate debug, test, trace, and replay activities across the entire distributed system without the use of any auxiliary interface or special cabling?

A NEW SOLUTION FOR DISTRIBUTED DEBUGGING

In the following example, a distributed system is assumed that contains a plurality of nodes, where every node is a self-contained processing unit with peripherals or, in other words, an embedded system. The nodes are connected via a network (for example, Ethernet) and exchange data to jointly perform their application tasks. No restrictions shall apply to the underlying network technology, be it wireless or cable-bound, shared, or switched. Today, such setups are deployed in automotive applications, industrial, and building automation, as well as machine and plant control, to name a few.

As already mentioned, a global notion of time is a key element for distributed debugging. Therefore the proposed solution for distributed debugging implements a mechanism to synchronize the local clocks contained in the network nodes. The clock synchronization mechanism can be implemented either in software or in hardware. The latter option, however, provides a better accuracy. If, for example, the IEEE 1588 clock synchronization standard is applied to a 100 Mbit/s Ethernet network, the clock synchronization mechanism can be implemented in hardware right above the physical layer, which allows

Comparison between a traditional approach to debug distributed systems (a) and the proposed new solution (b).

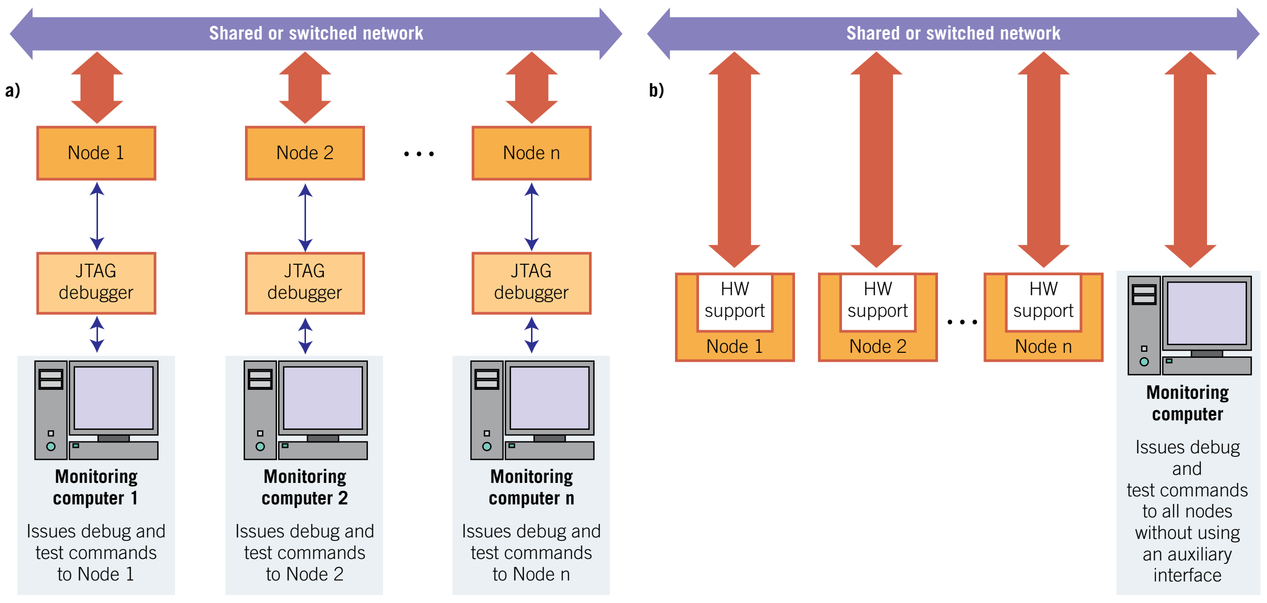

Shared or switched network
a)
Node 1
Node 2
Node n
JTAG debugger
JTAG debugger
JTAG debugger
Monitoring computer 1
Issues debug and test commands to Node 1
Monitoring computer 2
Issues debug and test commands to Node 2
Monitoring computer n
Issues debug and test commands to Node n
Shared or switched network
b)
HW support
Node 1
HW support
Node 2
HW support
Node n
Monitoring computer
Issues debug and test commands to all nodes without using an auxiliary interface

Figure 1

synchronization of the local clocks with a precision of about 10 ns. This enables systemwide debugging of multiple nodes at the CPU instruction level, assuming CPU clock rates up to 100 MHz. If the network provides an implicit clock synchronization mechanism, such as time-triggered protocols like TTP or FlexRay, this clock can be used for debugging purposes.

Besides the clock synchronization mechanism, the second key element in the proposed approach for distributed debugging is the usage of the already existing network to transfer debugging data. During normal operation, the network does not exhibit any additional traffic. Even the clock synchronization, although typically not showing higher traffic than one short message per second (in case of IEEE 1588), can be turned off if not needed. The nodes jointly perform their application tasks (its normal operation) without the need of interaction. To enable debugging and testing, a debugging master or monitoring computer running a debugging program, has to be connected to the network. First, clock synchronization will be activated to enable the common notion of time. During a debugging session, the debugging master transfers messages containing debugging commands to one or multiple nodes. Each node responds to the received messages and performs the intended tasks.

In an integrated approach, dedicated hardware units are added to each node that facilitate test, replay, monitoring, fault injection, or debug actions in the target embedded system (see **Figure 2**). All these local actions are controlled by an offload-engine without having the node's application CPUs to run a single line of additional debugging or test-related code.

Although the proposed solution is aimed for an implementation on hardware level, it's also possible to implement it in software. This would allow a less expensive implementation

! This enables complex test or debug scenarios where the sequence of events can be independent of network packet delays even if the nodes are distributed.

as no special chip has to be designed but comes with some restrictions and drawbacks. The major drawback is the intrusiveness of the software approach since the debugging software task can significantly change the system behavior.

BENEFITS OF THE SOLUTION

An unprecedented coordination of the plurality of nodes in the distributed system is reached by deriving the triggers for trace, debugging, or any other activities from the synchronized clocks as well as to enable cross-triggering between the hardware support units of the nodes. This enables complex test or debug scenarios where the sequence of events can be made independent of network packet delays even if the nodes are distributed in space.

An informal collection of some possible scenarios that would greatly disburden the task of testing and debugging in a distributed system could mention:

- Start and stop of code execution
- Activation of breakpoints
- Trace and display of register contents
- Trace of internal bus activity
- Synchronous replay (such as previously recorded data from sensor interfaces)
- Single stepping
- Start and stop of online as well as offline tests or maintenance activities
- Injection of faults
- Precise performance analysis

These actions can be executed on a single node, multiple nodes, or all nodes of the distributed system in a coordinated manner.

IMPLEMENTATION DETAILS

In the proposed solution several units are to be added to an embedded network node, typically and most effectively on chip level, to keep the task of debugging in the background and to minimize or cancel the probe effect. This location is also best for the clock synchronization mechanism. A node constructed according to the presented new solution thus contains a clock unit, a test unit, a replay unit, a monitoring unit, a fault injection unit, a debug unit, and an offload-engine, which controls the aforementioned units. The units are connected to the offload-engine via a dedicated on-chip debug bus, which is separated by the system bus for the application CPUs. Furthermore, a network interface is connected to the application CPUs (see **Figure 2**).

The network interface has a built-in filter that detects incoming messages that contain debugging commands or clock synchronization information and directs those messages to the offload engine. Other messages are forwarded to the application CPUs, which are unaware of the filtering mechanism. Regarding transmission the network interface has the ability to insert packets containing information related to debug and test from the offload engine into the transmit buffers during idle times of the network.

The offload engine comprises the subsystem that is responsible for processing messages for clock synchronization and debugging. Its task is to read incoming debugging and clock synchronization packets and to transfer the information to the corresponding unit (clock, test, replay, monitoring, fault injection, or debug unit). The performance of the offload engine can be significantly lower than the performance of the application CPUs since the task of forwarding messages to the units on the debug bus is relatively simple and the clock synchronization calculations are only repeated at a low frequency.

The test unit's task is to perform self-test operations in the node and to offer a network-enabled test access port to the system, for example by implementing JTAG ports to connect to off-chip components. That way, a device contained on a node could be configured through the TAP-master in the test unit, which is, for example, helpful for maintenance purposes.

To reproduce complex debugging setups, it might be useful to replay certain data streams from peripheral units. The task of the replay unit is to record data from external units such as GPIOs (general purpose input/outputs), UARTs (universal asynchronous receiver/transmitters), and DACs (digital-to-analog converters) and to allow playback at later times.

The monitoring unit is connected to the system bus as a passive listener and tracer. It records all or a filtered subset of accesses to the main system bus to allow a detailed history of transactions to be transferred to the debugging master for off-line debugging.

! A precisely coordinated approach to debug and test such systems [real-time video imaging applications] is as innovative as promising.

Fault injection is a method to assess system capabilities like fault detection, fault isolation, or recovery. In software testing, fault injection is used to test rarely reached parts of code. If required, such functions can be supported and as a consequence coordinated across the entire distributed embedded system.

The debug unit is designed to connect to the debugging interfaces of the application CPUs. The unit's functions are for instance to halt the processors of one chip, to single step through instructions, to add and remove breakpoints, and to allow access to processor internal registers. The use of a highly synchronized time base from the clock unit allows the debugging operator computer to perform a coordinated halt or single step of the complete distributed system or a remote reset after a nonrecoverable crash of the node.

The application CPUs contained in the nodes are the processing units that perform the actual application task. The (single or multiple) processors are connected to peripheral units like sensors and actuators as well as to the communication controller. Due to the nonintrusiveness of the debugging functionality, the application CPUs are unaware of any extra load caused by commands sent from or received by the debugging operator computer.

The software running on the operator computer is a crucial part, since the hardware support alone does not make much sense without any user interface to operate and control debugging and testing of the distributed system. Currently, the operator software is based on the well known tools GDB (GNU Project Debugger) and Eclipse to offer full state-of-the-art source-level debugging support for all nodes of a distributed system according to the solution we present in this article.

APPLICATION SCENARIOS

With respect to both CPU clock speed and data throughput, real-time video imaging applications where several high-speed, high-resolution cameras operating in an ensemble, interconnected via 10 Gbit/s Ethernet, are among the most challenging systems to debug. A precisely coordinated ap-

Overview of a distributed system based on the proposed solution for distributed debugging and details of the hardware support built into the embedded nodes.

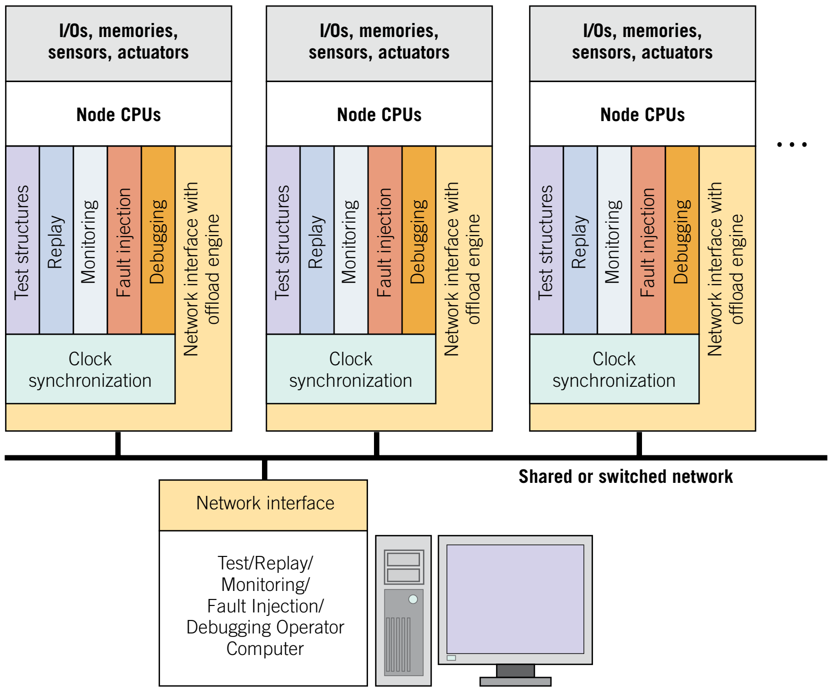

I/Os, memories, sensors, actuators
Node CPUs
Test structures
Replay
Monitoring
Fault injection
Debugging
Network interface with offload engine
Clock synchronization
I/Os, memories, sensors, actuators
Node CPUs
Test structures
Replay
Monitoring
Fault injection
Debugging
Network interface with offload engine
Clock synchronization
I/Os, memories, sensors, actuators
Node CPUs
Test structures
Replay
Monitoring
Fault injection
Debugging
Network interface with offload engine
Clock synchronization
...
Shared or switched network
Network interface
Test/Replay/ Monitoring/ Fault Injection/ Debugging Operator Computer

Figure 2

proach to debug and test such systems is as innovative as promising.

In telecom applications, network interconnect is increasingly moving from circuit switched or time division to internet-protocol–based solutions often relying on Ethernet network technology. At the same time, requirements for precisely retaining systemwide 2.048-MHz or 4.096-MHz sampling rates have to be fulfilled. IEEE 1588 clock synchronization is therefore currently built into such systems in many places. Including IEEE 1588 would, at the same time, enable corresponding coordination of test and debugging support if taken into account.

Distributed systems are also rampant in industrial automation. Here, real time is often defined with respect to time constants of mechanical systems such as motors, which typically are in the range of several 1 µs. As Gigabit network bandwidth is also beginning to break into this area, CPU clock speeds and application complexity are very likely to be increased accordingly.

Automotive systems have just started to step to higher bandwidth—for example, FlexRay communication technology with 10 Mbit/s data rate. The predicted shift from federated toward integrated architectures will very likely further increase complexity in those systems, as multiprocessor break-by-wire chips are already being designed both due to reliability and performance reasons. A systematic approach to debugging and test, precisely coordinated amongst the collaborating embedded control units promises to greatly ease efforts during system bring-up as well as maintenance.

UNPRECEDENTED INSIGHT

Testing and debugging of a distributed system is known as a complex task. The presented solution enables the developer to gain control over all the nodes in a distributed system in a coordinated, reproducible, and, if required, a nonintrusive manner. The main difference to existing approaches is the fact that the proposed one takes the distributed character of the system into account. The combination of precisely synchronized clocks with hardware support for test, replay, fault-injection, monitoring, and debugging allows for unprecedented insights in the execution flow of distributed systems. ■

Roland Höller and Peter Rössler are responsible for R&D projects at the University of Applied Sciences Technikum Wien, Austria. They've worked many years in the area of ESL design, FPGA and digital ASIC design, PCB design as well as clock synchronization and control networks. The work described herein receives support by the City of Vienna, Department MA27 (grant numbers MA27-Project 04-11 and MA27-Project 05-05) and the Austrian Research Agency FFG (grant number 818647).

Connect with the leader in embedded USB software.

Micro Digital provides integrated USB solutions that run out of the box with our SMX® RTOS, file system, and TCP/IP stack. These robust, high-performance USB solutions are written in ANSI-C, and can run on any hardware platform, with SMX, another RTOS, or stand-alone. Connect with us today at **www.smxrtos.com/usb**.

Connect to:
- USB disk drives
- USB serial devices
- USB keybds, mice, HIDs
- USB audio devices
- USB modems
- USB printers
- USB-to-Ethernet adapters
- USB-to-serial adapters
- USB-to-WiFi w/WEP & WPA

Look like to a PC:
- USB disk drive
- USB serial device
- USB keyboard & mouse
- Audio with MIDI
- Ethernet over USB (RNDIS)
- Composite devices
- Multi-port serial

USB Controllers Supported:
OHCI, UHCI, EHCI, OTG
Atmel AT91 • Cirrus EP
ColdFire • Freescale i.MX
Luminary LM3S • Maxim 3421
NXP ISP, LPC • Sharp LH7
ST Micro STR7, 9, STM32

Features:
USB 2.0 compliant • Low, full & high speed • All transfer modes
Small & fast • Easily portable
Standalone operation • Royalty-free cap licensing • Full Source Code

μd Micro Digital
USB SOFTWARE LEADER
800.366.2491 sales@smxrtos.com

www.smxrtos.com/usb

EE TIMES ACE AWARDS 2010

Tuesday, April 27, 2010

Presented at the Embedded Systems Conference Silicon Valley

CALL FOR ENTRIES

2010 EETIMES ACE AWARDS

EE Times presents The 6th Annual EETimes ACE Awards

The Annual Creativity in Electronics (ACE) Awards celebrates the creators of technology who demonstrate leadership and innovation in the global electronics industry and shape the world we live in.

If you or your company has made significant achievements in 2009, enter today to see if you can become part of a prestigious group of finalists and winners recognized by EE Times editors, a distinguished judging panel and the global electronics industry.

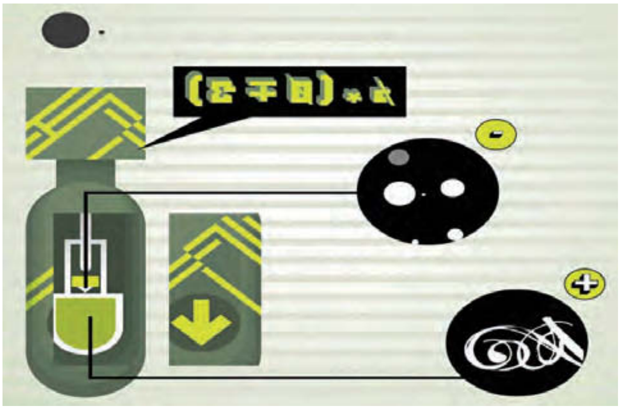
CATEGORIES

- DESIGN TEAM OF THE YEAR
- INNOVATOR OF THE YEAR
- EXECUTIVE OF THE YEAR
- STARTUP OF THE YEAR
- COMPANY OF THE YEAR
- MOST PROMISING NEW TECHNOLOGY
- EDUCATOR OF THE YEAR
- STUDENT OF THE YEAR
- MOST PROMISING RENEWABLE ENERGY AWARD

To enter, go to www.eetimes.com/ace

Entry Deadline – January 22, 2010
No cost to enter

EE Times



GUI testing traditionally meant finding the most appropriate access point to inject test cases. The challenge, however, is in making the GUI tests repeatable. Here's a homegrown framework that allows test input to be managed, replacing manual test cases.

GUI testing—exposing visual bugs

BY NIALL MURPHY

My approach to testing a graphical user interface (GUI) has always been to find the most appropriate access point to manually inject test cases. This article will discuss the challenges of trying to make GUI tests repeatable, and we'll look at a homegrown framework that allows test input to be managed.

To test a GUI, we need a framework that gives us the ability to inject test cases easily and to observe the output. Ideally, in a test system, the output can be stored, and then a subsequent test run can be compared with a previous run to provide regression testing. By regression testing, we mean that we have proven that the new version under test has not broken anything that used to work for the previous version.

If we consider a non-GUI issue like message processing, you might introduce a test set of messages and then the application's message store, or inbox, will contain those messages. If the inbox from one run is compared with the next, there are two possible outcomes. The inboxes may be identical—this means that the new functionality has not broken previous features. In some cases, you might have expected changes to result from new functionality in the system under test—in other words, the output should have changed but didn't, telling us that the new features are not working.

The second possibility is that the outputs are different. If the inbox messages can be expressed in text format, some text difference tool can show those inconsistencies and the engineer can examine them to decide if the changes match the new functionality. For example if an urgency field has been added to the messages, a message identifier might change from:

```
Message index:17,
message title: test_185_title
```

to:

```
Message index:17,
message title: test_185_title,
urgency: normal.
```

If the only changes in the output are consistent with this, we have successfully added the urgency feature, and we have not broken anything else.

It would be great to apply the same principle to graphics, but the challenge is that the appearance of the display can not usually be expressed in a humanly readable text format. This means that it is tricky to examine differences. The output of a GUI test is the appearance of the screen, which is a large number of pixels, and each pixel has a color value. While a human being can view the screen and establish if the text is readable and the layout conforms to requirements, the job of checking if each pixel has the correct numerical value (or color) is a nontrivial task.

A screen-capture tool could record the screen and a comparison tool could simply point out if the screenshot has changed, and then allow the tester to view the old and new image. In theory this is a good approach, but it can be very labor intensive. The reason is that the number of screenshots for a GUI can be very high. Usually it is desirable to capture the screen after every user event (such as button press), or external event (some alarm condition occurred). Unfortunately one global change can cause a change to all of those screenshots. For example, changing the default font, or background color, or making the margin at the left of the screen a little wider, will cause a change in every screenshot that has been gathered.

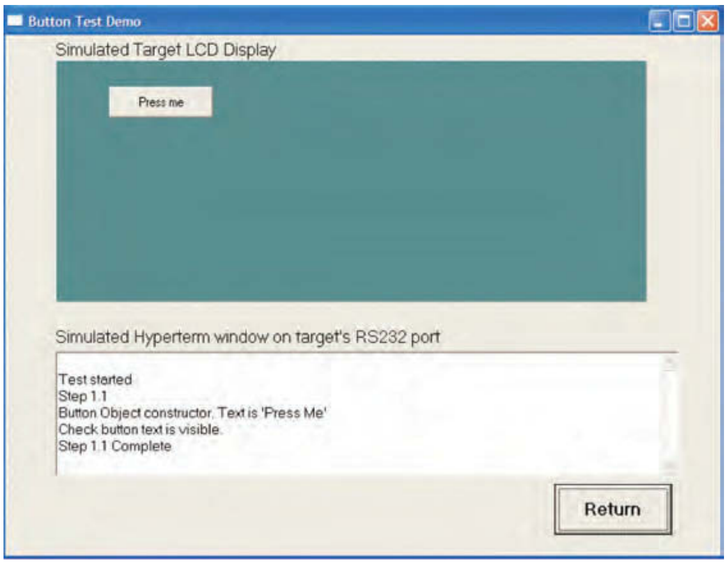
These are the sort of scenarios that often expose bugs, but exact sequence or timing requirements may be impossible for a human to reproduce.

Commercial tools are available to help automate the process just described, and while I would not discourage their use, be aware that in this context, automated testing does not mean labor-free testing. The other restriction of these tools is that they need access to the GUI's frame buffer, so a certain amount of integration between your system and the test tool is required. The amount of effort required to get the tool up and running will depend on your operating system and hardware platform.

At the other extreme, all tests could be manual, where the test document instructs the tester to press certain buttons in a certain order, and then observe the results. While this approach does not require any code to be written, it is extremely weak for a number of reasons. One is that it is very time consuming for the tester to read instructions before each button press. If the tester makes a mistake, he will most likely have to restart a sequence. At the end of a sequence, if the output is not correct the tester will be left wondering if the test failed because of a bug or because they might have pressed the wrong button without realizing it, so they may run the sequence again just to be sure.

An even bigger disadvantage is that you can not call specific functions using the method above. In some cases you may want the tester to press on a button, then simulate some external event that causes an alarm to appear on the display and then get the tester to release the on-screen button. These are the sort

The first step of the test creates a button.


Button Test Demo
Simulated Target LCD Display
Press me
Simulated Hyperterm window on target's RS232 port
Return

```
Test started
Step 1.1
Button Object constructor. Text is 'Press Me'
Check button text is visible.
Step 1.1 Complete
```

Figure 1

of scenarios that often expose bugs. But scenarios that have exact sequence or timing requirements may be impossible for a human to reproduce.

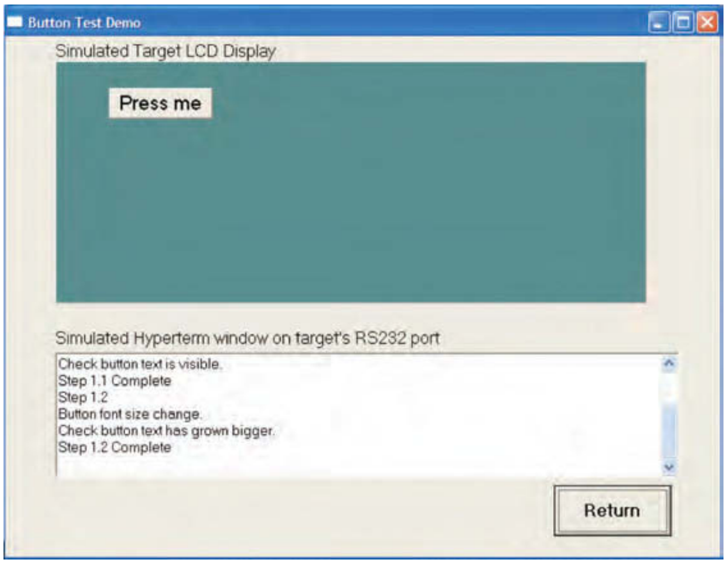
Similarly if the test requires that an on-screen event happens at an exact x,y location, the tester cannot guarantee that that acted on the exact required location, especially on a touchscreen with no visible mouse.

Finally, testing the system with no access to the code means that the test can not access internal data structures at the end of the test to check if they have the correct values. The tester might be able to observe on-screen or external state, but they have no visibility of the state changes internal to the software that occurred during the test.

ROLL YOUR OWN

A simple test framework can exercise test cases in your GUI. These test cases

The second step of the test suite shows the font size being changed.


Button Test Demo
Simulated Target LCD Display
Press me
Simulated Hyperterm window on target's RS232 port
Return

```
Check button text is visible.
Step 1.1 Complete
Step 1.2
Button font size change.
Check button text has grown bigger.
Step 1.2 Complete
```

Figure 2

”We are excited...

...that Hansoft can handle both schedule driven projects and agile methods like Scrum. The possibility of combining these methods in the same project has really been useful. Hansoft makes our projects more efficient and helps us collaborate much better with partners, customers and suppliers.”

General Manager,
Space Systems, Swedish Space Corporation.

HANSOFT

Download a free 2-user solution trial at www.hansoft.se can not be fully automated, because a human observer must confirm the appearance of the GUI. They allow the tester to run them in a reasonably efficient fashion, and they make it straightforward to ensure that one test run is consistent with the previous run.

There are two levels of testing that are of concern when you build a GUI. One is the low-level graphical operations accessible via a function call, for example drawing a line in a particular color. The second type of testing is where you want to simulate events that occur on the finished product. Much of this involves simulating mouse/touch events on the screen, or simulating external events such as changes to the analog or digital input lines. We will look at each of these situations in turn.

TESTING LOW-LEVEL GRAPHICS

In many modern GUIs, the low-level graphics primitives are provided by a graphics library, so the issue of testing this portion of the software may already be taken care of by the company providing the library. Even if you are using a well-established product, you may choose to write tests for some portions of it.

Of course, if you are creating and selling a library, then you need a test infrastructure to test the whole library, and being able to repeat the tests will be important at each release.

At *www.panelsoft.com/GUItesting.htm* you find an executable that runs a few simple tests of a button object. The executable runs on a Windows PC, and simulates the type of testing you can perform on an embedded GUI, using an RS232 port as a means of getting test information out of the system. Each step of the test performs some small change on the GUI and either the test code or the tester checks that the appropriate change has happened.

For almost all graphics projects, there are huge advantages to making the system work on a desktop computer as well as on the target. All commercial embedded GUI libraries support this arrangement. This applies to the final product code, but it also applies to the test code. The tests can be developed far more quickly on the PC with a simulated display. The completed tests are then run on the target system, which will uncover any problems that arise on the target that were not an issue on the PC. The PC simulation and the target display should be pixel-for-pixel identical in theory, but you can still get bugs that occur on the target, but not on the PC. Problems with running out of memory or timing issues are possible sources of trouble. Also the PC is likely to be driven by a mouse, while the target may use a touchscreen. This difference might disguise some problems on the PC.

! For almost all graphics projects, there are huge advantages to making the system work on a desktop computer as well as on the target.

STEP BY STEP

The demo test does a few very simple checks on the functionality of the button object in the system under development, but for the purposes of this example, we are not concerned with whether the button is being fully tested. We are more concerned with exploring the test harness that makes it possible to manage lists of these tests. Once the harness is in place, it provides a home for tests to be added, for new features, or in response to bugs.

How the harness is coded will depend on the communications mechanism available, but a serial port is typical. At each step some test code is run and the tester is prompted to check the display for certain properties. The example test code tests a button object, and **Figure 1** shows the state at the end of the first step, which simply tests the construction of a button object.

A step like this requires the tester to confirm that the button is visible and its text is readable. The test code will often check internal values. For example a call to the button's `getText()` function could be checked to see if it returned the string "Press me". The advantage of these checks is that they do not require any human interaction and so do not add to the test time.

Pressing 'Return' on the RS232 interface (probably using a terminal emulator program on a PC) will advance to the next step. **Figure 2** shows this display after the tester has advanced to the next step. This step modified the font used in the object that has already been created. Changing the appearance in each small step allows the tester to observe if an operation that changes the object has the desired effect.

The level of detail of the instructions given to the tester will vary. One of the tricky things to check is the coordinate system. If this test places the button at position 52, 26, it's desirable to measure the distance from the top, or left, of the display to the button. Even if the pixels were large enough to be individually counted, checking 52 of them would be tedious and error prone. One approach is to get the test code to draw a horizontal line 52 pixels from the top of the screen and then observe if the line is aligned with the edge of the object under test, as shown in **Figure 3**.

Of course drawing guidelines assumes that the lines will come out in the correct position. So this test would

Guidelines confirm the button is in the correct position.

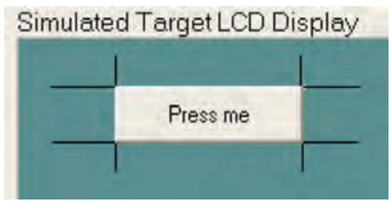


Figure 3

catch a positioning bug that is specific to the button, but it would not catch the case where all x positions were incorrect by 3 pixels. In practice, you will want to do some position checks, but it is too labor intensive to add guidelines for every positioned object, and so once the coordinate system is tested enough to be considered trustworthy, the testing effort can move elsewhere.

If the test is initially run on a PC, a number of options are available, that might not be possible on the target. One is to do a screen capture and paste the screen into a drawing tool. Most drawing tools will report the position of the cursor as an x,y position and so you can measure the distance, in pixels, between any two points in the captured screenshot. **Figure 4** shows an example using Paint Shop Pro.

Once you have done a screen capture, you also have the chance to zoom in to examine the details. This is especially useful if the screen is of high resolution and subtle details, like anti-aliasing at the edge of letters can not be easily seen with the naked (and sometimes aging) eye. **Figure 5** shows how zooming in allows us an up close view of the anti-aliasing being applied. Part (a) shows no anti-aliasing. When this is rectified, part (b) shows the edges being anti-aliased. The color choices for the softening of the edges are not ideal, and this is due a limited color palette. If this is rectified in software, the changes to the edges of the letters would be quite subtle, and zooming would be vital to examine the change.

A typical PC simulation copies the target screen pixel for pixel. On some projects, I have modified the simulation to allow an option of doubling the number of pixels in the horizontal and vertical directions. This effectively is a 200% zoomed view of the target and can be very useful when examining the details.

CODE ORGANIZATION

A good test harness must make it easy to insert and remove tests. Each set of tests should be runnable independent of the last set, to allow them to be run in reasonable size chunks. For example, all of the button tests might be built as one executable, and so they can be run completely independent of the slider tests. This means that a problem with the button tests will not impact the slider tests. Within the suite of tests for button, sectioning the numbers as 1.1, 1.2, 1.3, and so forth, and then another section as 2.1, 2.2, etc., will allow a new test to be added without having to reorder every following test. In theory you could have used the rule that new tests must always be added at the end, but in practice, previous tests often set up the right state in which you can best run the new test, so the best place for a new test might be just after some closely related test that has set the right conditions.

Previous tests often set up the right state . . . the best place for a new test might be after some closely related test that has the right conditions.

There will often be a bit of code that must be run before or after each test—sometimes a function to refresh the display, or to flush all pending events is required. Also, during development, you want a lot of control over which tests are run and in what order. For example, if I am having trouble with test 4.16, I may want to run that test several times, making alterations each time, but I do not want to run every preceding test each time. To achieve this, I want to temporarily disable all the tests before 4.16. In some cases, I will want to keep any tests that set up the conditions required for test 4.16.

These requirements lend themselves to a structure where each test

The dimensions of the selection box are shown in the bottom border, which tells us that the top left corner of the button is in position 52, 26 on the simulated screen.

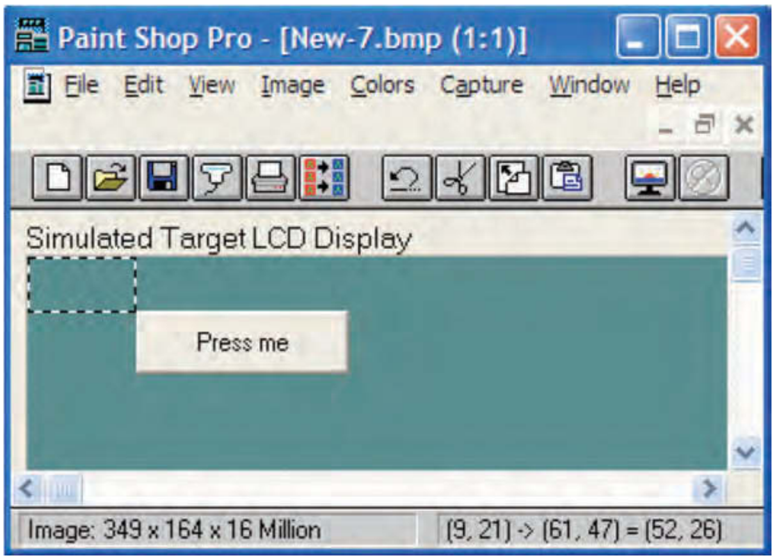

Paint Shop Pro - [New-7.bmp (1:1)]
File Edit View Image Colors Capture Window Help
Simulated Target LCD Display
Press me
Image: 349 x 164 x 16 Million
(9, 21) -> (61, 47) = (52, 26)

Figure 4

step is a function and a table of pointers to functions dictates the order in which they will be called. Commenting out some portions of the table allows a bunch of tests to be temporarily disabled. Part of the array might look like **Listing 1**.

Each function name in the list represents a step of the test. Note that the ButtonClick part requires two entries in the table. This is because the harness progresses to the next step each time the tester presses return and some tests require more than one press of the return key to check all parts. In this case there is a setup step and a check-at-end step. Most of the time, it is a purely semantic issue whether such setup and check pairs are considered to be two parts of one test or whether they are considered to be two tests that have a dependency on each other.

EVENT CHECKING

In the case of the button test, the reason two steps are required is that at the first step, the tester is prompted to click on (or touch) the button, and then on the second step the test code confirms if the tester did in fact click on the button. For this to work, the first step has to set up the button's event handlers in such a way that the click is recorded in some boolean flag. In the second step, the flag is checked. If it is still false, the event never registered. Either the tester did not follow the instructions, or the event handling has a bug.

If you run the demo executable, and if you do not click on the button when instructed, the next step will print a failure message. Because the event handler also prints a message to the serial port, the tester could have just observed that there was a response to the event. However checking a flag means that there is less reliance on the tester and therefore less opportunity for human error.

Creating false button presses is sometimes more challenging that you would think. If there are 10 buttons on the display, the test code may not have pointers to those objects.

A similar approach should be taken to any other objects that the tester manipulates. If the tester is instructed to move a slider to its maximum, the tester can be instructed to check the on-screen value of the slider, and the test code can also query the internal stored value. Both checks are necessary to ensure the object is storing and displaying its value correctly.

THE BIG PICTURE

Most of the discussion so far has assumed that individual graphical objects are being tested. In many cases, the underlying objects are trustworthy, but the application logic needs to be tested. I generally use the same test harness to test the application, but instead of calling individual functions of the object's interface, I fake mouse/touch events or eternal events that have an impact on the appearance of the interface, such as an alarm. Other external stimulation might be a varying analog signal that is being graphed on the GUI.

Simulating the external data sometimes means replacing a function such as `readAnalog()` function with code that accesses a table of test data instead of the analog hardware.

Creating false button presses is sometimes more challenging than you would think. If there are 10 buttons on the display, the test code may not have pointers to those objects. The only pointer to them might be inside the window, or group, object that contains them, and since that window contains many other objects, it might not be trivial to query the window for the specific button that you require.

In some cases, the creation order of the objects is known. For example, the test author might know that the fifth object created within a window is the button the required for the test. So the test code iterates through the list of buttons contained by the window until

Zooming in on (a) shows that antialiasing is not being used, and (b) shows that it is.

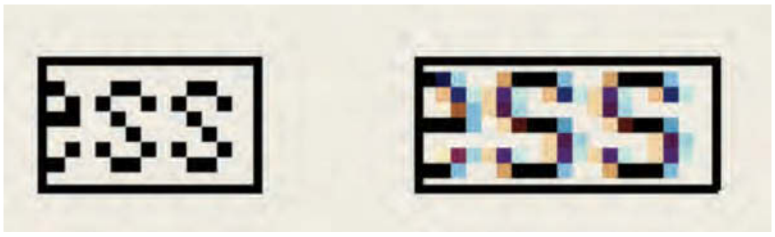


Figure 5

Listing 1

```
typedef void (*TestFn)(void);
TestFn TestList[] = {
TestCreateButton,
TestFontSize,
TestButtonClick1,
TestButtonClick2,
NULL
};
```

it reaches the fifth one. This approach is not entirely satisfactory, since minor changes to the order of construction will break the test.

Some libraries will allow you to seek a button without knowing the pointer. For example the PEG library allow an identifier, which is an integer, to be associated with any object. A `find( )` function is available in the top level window, which will recursively search all windows to locate the object with the a matching identifier. This is better than the previous approach, but still has weaknesses. One is that the identifiers are not necessarily unique, so two buttons in two different windows might have the same identifier. The second problem is that it only locates objects that are on the display. The test code may need to access an object before it becomes visible, in order to alter some of its properties.

Another approach that doesn't require pointers to the button objects, is to generate mouse/touch events at a low level, and the x, y, position of those events is set to correspond to the position of the required button. This has the drawback that changing the position of a button means that the test might not work as expected.

While I would not recommend this method for all testing, there are some cases where generating low-level mouse/touch events are ideal. If you want to test what happens when you touch the edge of the button, specifying the x,y position allows you to simulate a user's touch on an exact location that sits on the button's border. Note that this is something that would be impossible for a human tester to do on a touchscreen, since the human finger is just not that accurate. Another example of a test that suits this method is where you simulate a press-down event inside the button and a release event outside the button.

If you plan for it early in the development cycle, the buttons and other objects required for testing can be registered with the test harness, using some enumeration type to index them in a table, which will make them always accessible to the test code. This registration step is often conditionally compiled code within the application itself. Adding code to the application in order to allow the test harness to work is not ideal, since it can make the build process more complex, but in this case it may be justified if it means that we get a clean mechanism for faking button events.

! This has the drawback that changing the position of a button means that the test might not work as expected.

At this point you might be thinking that faking button presses might be more trouble than it is worth. If the tester has to press return at the end of each step, why not let the tester press the button on the GUI and avoid the need to generate button events from the test code. In practice, it would be very difficult to get the tester to follow an exact sequence of button presses. In many cases the test may require multiple buttons for one step—the intermediate states in the GUI may already have been tested, so the goal is to check the final state that resulted from many user events.

SHOWING ALL STRINGS

A test suite that displays all test strings is very useful, especially if the GUI is going to be translated into foreign languages. Each step of the test navigates to a different state, or changes some

Part (a) looks fine but changing the background colors in (b) reveals that the address field is too wide and overlaps with the neighboring icon.

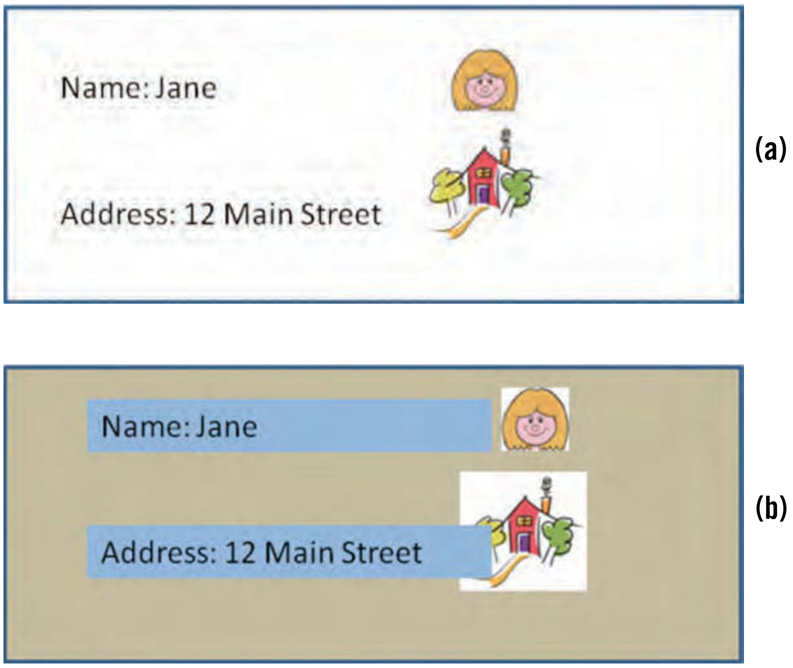
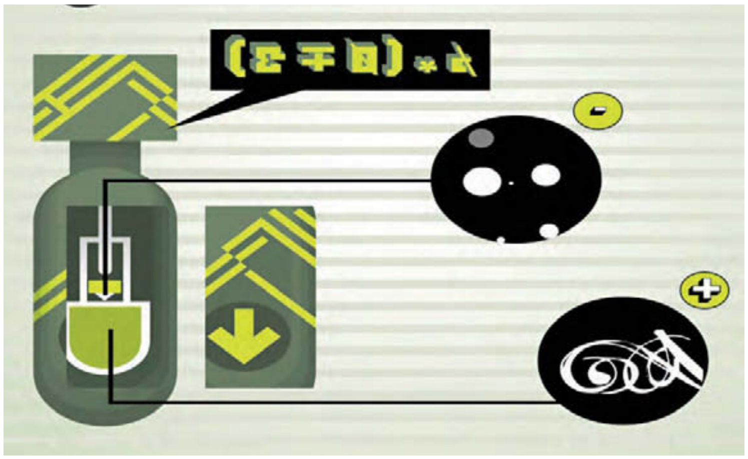


Figure 6

external condition to display a string that has not been seen before. Of course, there are many strings that get seen at multiple steps of the test. The 'OK' string in one of your buttons might be visible at almost every test step. The serial port output of each step should identify which of the strings on the display have not already been viewed on a previous step.

Once you have a test that displays all strings, you can be confident that you will catch any strings that are too long to fit in the space allocated to them on the screen. When the product has been translated into a foreign language, the same test can be run to ensure that the translated strings do not overflow the areas that were sufficient in the English version.

If all translated strings can be generated in a simple list, I usually print out that list and tick off each string as it is viewed in the test, so that at the end, I can tell at a glance if there is a translated string that did not get tested.



Any windows that were opened should be closed. Any alarms that were raised should be cleared. At this point, the heap size should be exactly the same as it was at the start. If not, you should suspect a leak.

MEMORY MANAGEMENT

Some GUI libraries use the heap and some do not. A major concern is whether use of the GUI library leads to any memory leaks. Letting a GUI run for days, and checking memory consumption occasionally, does not really stress the heap. If there is a memory leak, it is caused by the response to a particular event. So the key to testing the heap is to measure heap size and drive a sequence of events, and then at the end of that sequence, check if the heap has grown.

So the memory management test should start at some neutral state, where there are no outstanding events on the GUI. Then the test code should navigate through every screen, trigger every conceivable event, and then navigate back to its starting position. The external events may be things like alarms that cause a warning to popup on the display, or anything else that leads to GUI interaction.

When the test code arrives back at the starting state, there should be no partially processed events. Any windows that were opened should be closed. Any alarms that were raised should be cleared. At this point, the heap size should be exactly the same as it was at the start. If not, you should suspect a leak.

Be aware that in some designs objects will consume space the first time they are used. For example, the first time an alarm occurs, space might be allocated for the alarm message. All following occurrences then use that same piece of storage. So one sequence through the test will result in a heap that is bigger at the end than it was at the start. For this reason, I prefer to run this sequence twice. I ignore any heap growth on the first run of the test, but the second run should not cause any further growth.

If there is growth in the heap, this test may detect it, but it does not identify the exact cause. There is further discussion of how to pinpoint the cause of a memory leak in my "More on Memory Leaks" article.[1]

LOGGING AND PACING

Many types of applications lend themselves to test-by-log-file, where a test is stimulated and as the application performs actions, they are also logged to a file, or transmitted on a serial port. Generating the log file depends on the application including a logging feature, which might be conditionally compiled. Following the test, examination of the log file will indicate if the correct actions took place. This is particularly useful if the actions are difficult to observe externally. On a GUI, by definition, almost everything is visible externally and log files are not used very much.

There are a few cases where I have found logging revealed things that would have been difficult to spot on the display. A log file that records each item drawn on the display makes it possible to see the order in which they were drawn, and also if an item was drawn twice. It is not unusual to have a bug in a GUI that leads to part of the display being updated twice. Such a bug is relatively harmless, but it is a waste of CPU cycles to draw the same items more than once. Since the final state of the GUI is the same, a tester might not spot this. However if a log file is examined and it states that certain items were drawn more than once then the problem can be identified.

Another way to detect the same style of problem is to deliberately slow down the GUI. Add a delay in some low-level drawing routine and the whole GUI will move in slow motion. You will see the display being gradually built up as different elements are drawn. If an item is drawn, then erased, and then drawn again, you will see this happen.

Another common flaw that is exposed by this technique is where one component is drawn but then completely covered by some overlapping objects. Since there was no benefit in drawing the elements that get covered, the software could be optimized to remove that step.

A second method for slowing down the GUI is to disable the clock that drives the on-screen events. Each time the tester presses return, the clock is advanced a few ticks. I have found this very useful when testing waveforms that are tracking some analog input, such as an oscilloscope display. Running at full speed, it is difficult to observe the changes from one tick to the next. By freezing the clock, the waveform can be observed as it advances from one sample to the next on each press of the return key.

BORDER CROSSINGS

The alignment of objects and whether they overlap is often only visible if the borders of the object are visible. In many cases those borders are deliberately invisible. For example a piece of text will often not show its bounding box. A bitmap can use the background color so that it floats on the background instead of appearing as a rectangle.

For test purposes, seeing those boundaries is important if we want to ensure objects do not overlap. Fields that vary in length, such as names and addresses entered by the user, might look OK when the data is short, but cause problems if the field is filled completely. **Figure 6** illustrates this as one of the text fields will overlap with an icon if the text field is completely filled.

Changing a few of the background colors in a test build will often make these issues visible. You do not want to scatter conditionally compiled background definitions throughout your code. However, if all of your color definitions are arranged in one header file, changing the definition of the most common window background may show up most of the objects of interest.

! Running at full speed, it's difficult to observe the changes from one tick to the next. By freezing the clock, the waveform can be observed as it advances from one sample to the next on each press of the return key.

BUGS VERSUS TASTE

One of the challenges of interpreting test failures is distinguishing between outright bugs and issues that might be open to interpretation or taste, such as usability issues. One tester might consider a screen to be perfect while another tester might consider it difficult to read because of font size or difficult to understand because of poorly worded text. Other usability issues such as size or position of buttons may also be open to debate. These subjective failures may be rejected by developer on the grounds that the software was still meeting its explicit requirements.

It is important to have a process in place that allows these issues to be resolved. The process might state that no usability issues are to be addressed at the test phase, or it might state that the project leader will arbitrate if the tester and developer cannot agree if a particular test has failed, or you could hand all power to the tester and allow him to fail any test where he thinks the look of the GUI might not meet customer expectations. As long as the process is understood by all sides, it can avoid endless debate between developers and testers.

FIRST IMPRESSIONS, LASTING IMPRESSIONS

Testing the visual elements of your GUI is vital because the GUI forms the customer's first impression of the product and any quality issues in the GUI will be interpreted as quality issues for the whole product. Different areas of software will always require some tweaking of the test methods employed to maximize the number of bugs caught, and hopefully some of the techniques shown here will help you track down a few extra GUI bugs. ■

Niall Murphy has been designing user interfaces for over 14 years. He is the author of *Front Panel: Designing Software for Embedded User Interfaces*. Murphy teaches and consults on building better user interfaces. He welcomes feedback and can be reached at nmurphy@panelsoft.com. His web site is *www.panelsoft.com*.

ENDNOTES:

1. Murphy, Niall. "More on Memory Leaks," *Embedded Systems Programming*, April 2002, available on-line at *www.embedded.com/story/OEG20020321S0021*

Blackberry Storm has some good qualities, and some not so good

BY RICHARD NASS

The imaging and telephony functions were up to snuff. The UI left a little to be desired.

The BlackBerry Storm came out not too long after I purchased my Blackberry Curve. Like lots of the latest consumer electronics devices, you're always hesitant to jump in and make a purchase, because you know your device will be trumped by something cooler in a very short time.

That was the case with my Curve, but not exactly. Shortly after I acquired my Curve, the Storm crossed my desk, as the object of a Tear Down. My first inclination with a Tear Down is to try the device out for a while, to get a feel for the user experience. In this case, it was to see what I was missing out on.

To my surprise (and delight), I found out that the Storm doesn't have a better user experience. In fact, I like my Curve better. The biggest reason for that is the touch interface that accompanies the Storm. I found it quite difficult to press the right buttons. I may have big fingers, but they're not so big that I should have that much trouble with the interface.

Being the engineer that I am, I decided not to let that one design flaw completely ruin the user experience, although it was hard to get

passed that point. I found that as a telephone, the Storm worked really well. And that's an important component, although I use the phone more for e-mail and texting than I do as a phone (nobody calls anybody anymore, do they?).

The second feature I found to be quite attractive was the imaging quality, both for still images and for video. So that's where my Tear Down started. I wanted to find out what was the cause of those qualities that I found to be so appealing.

The sensor that's in the Blackberry Storm is from OmniVision Technologies, the OV3647. It's a low-power, low-cost CMOS image sensor. Containing a parallel interface as well as a mobile digital display interface (MDDI), the sensor has a resolution of 3 Mpixels running at 15 frames/s. It has embedded phase-locked loop (PLL) and can be embedded into a module that's 7 by 7 by 5 mm, as is the case in the Storm. It has an embedded 1.5-V regulator for operation at full power, but the core power is about 1.4 V.

The OV3647, which is related to the sensor in RIM's Blackberry Curve handset (the OV2640), supports a raw RGB output format, image sizes of QV-XGA and XGA, and has a programmable frame rate. Image quality controls like lens correction, auto exposure, auto gain, auto balance, and defect pixel canceling help enhance the images, along with support for LED and flash-strobe mode.

One of the unique features of the OV3647 is that it contains Qualcomm's proprietary MDDI. This is a key differentiator because it provides a proven solution for the interconnect challenge provided by the hinge in a flip (clamshell) phone. This reduces the wire cost, and ultimately reduces the overall system cost.

According to the engineers at OmniVision, the interface also helps reduce the EMI in the differential signaling protocol. This, in turn, eliminates the need to use a parallel port where you've got eight data bits swinging at 1.8 V with a clock and two syncs signals.

While the Storm obviously doesn't employ a clamshell design, the EMI is reduced nonetheless. Qualcomm supports MDDI on most of its newer chip sets. Hence, it gives OmniVision easier entry into a Qualcomm design.

The Blackberry Storm employs a Wideband CDMA HSPA UMTS power amplifier, the AWT6241, from Anadigics. This part allows the phone to be used in most regions throughout the world.

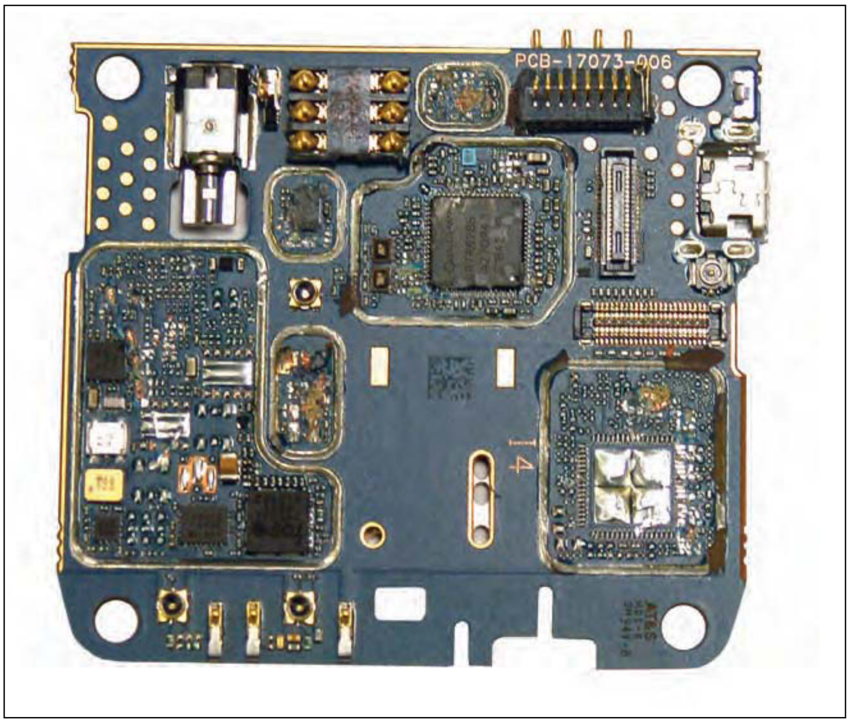


Figure 1

While the phone was designed by RIM, the designers at OmniVision were involved in integrating the sensor into the handset. This process occurred about 18 months ago.

While it's obviously a hardware integration, there were some software issues that OmniVision had to iron out with RIM, like which settings should be used. OmniVision's applications team worked hand in hand with RIM to get the best image quality, resolution, etc. That led to OmniVision doing some scripting for the Storm and tuning the image parameters, like the lens correction and the defect correction, so the image looks the best regardless of temperature. The OmniVision developers also had to ensure that they provided the modes for all image sizes.

THE MAKINGS OF A WORLD PHONE

The power amplifier (PA) in the Storm is a Wideband CDMA HSPA UMTS model, the AWT6241, from Anadigics. The device provides the amplification for the phone's 3G mode. This particular model is a world phone that runs on Verizon's CDMA network in the U.S. It enters roaming mode while in Europe, taking advantage of a GSM /EDGE power amplifier.

RIM chose the AWT6241 because of its high efficiency and high output power. The device is part of Anadigics' third-generation High-Efficiency, Low-Power (HELP3) product line. It's real claim to fame is its low quiescent current and the high efficiency at backed-off power. The result is a longer battery life, which is essential in a device like that Storm that contains a touch screen, a web browser, an e-mail client, and lots of other bells and whistles.

! Designers shouldn't get too hung up on saving components at the expense of good matching between the components and the RF subsystem.

"For the IMT band, the 6241 is compatible with the Qualcomm chip set which is really the heart and soul of the Storm," says Bruce Webber, director of marketing for Anadigics' wireless products. "Sometime next year, we'll have our fourth generation HELP parts, which offer even further improvements in efficiency and quiescent current."

Note that in the design of a handset, there are some shortcuts you can take, and others you should avoid. Handsets are obviously space-constrained, and designers would prefer to use a single-sided board rather than a double-sided board.

Webber says, "One of the things we've seen is that the continuing drive to reduce board area and BOM costs is causing people to leave out matching components. They'll try to do a one- or two-component match and sometimes that's not ideal. You need to leave yourself enough flexibility to match the power amp to the duplexer that's selected for best power output efficiency. Otherwise you can end up having problems with linearity or even oscillation under some circumstances."

Anadigics provides reference designs for many of its parts. And the company is quick to point out that designers shouldn't get too hung up on saving components at the expense of good matching between the components and the RF subsystem. Because that's where you can achieve the high efficiency, good linearity, and low spurious emissions, while simultaneously resulting in good RF performance.

"The end user might not be aware of all the stuff that goes on inside, but they certainly know when a call gets dropped," adds Webber. ■

PC Based Logic Analyzers

NEW Model DV3400 from $499.00 DigiView

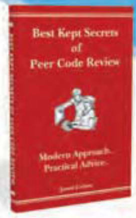
- 18 / 36 Channels
- Up to 400 MHz
- I²C, SPI, RS-232, included.

Professional Hardware Capture + Software Analysis

PICmicro® MCU Programmer only $199.00 QuickWriter™

TechTools www.tech-tools.com (972) 272-9392 • sales@tech-tools.com

FREE BOOK on Best Practices for Peer Code Review available at CodeReviewBook.com


Best Kept Secrets of Peer Code Review
Modern Approach. Practical Advice.

SmartBear SOFTWARE

Compact Embedded Server

The Compact SIB is a small, low cost, yet powerful Server that is ideal for hostile rugged environments & has the following features:

- Fanlessx861GHzCPU
- 256MBDDR2RAMOnBoard
- 128MBInternalFlashDisk
- 10/100Base-T Ethernet
- OnBoard Audio
- 2RS-232&3USB2.0Ports
- CompactFlash&MicroSDSlots
- Reliable(NoCPUFanorDiskDrive)
- OptionalWirelessLAN&HardDrive
- Dimensions:4.5"x4.5"x1.375"

2.6KERNEL

Windows CE

Pricing Starts at $230.00

http://www.emacinc.com/linux_compact_sib.htm

EMAC, inc. EQUIPMENT MONITOR AND CONTROL

Since 1985 OVER 24 YEARS OF SINGLE BOARD SOLUTIONS

Phone: (618) 529-4525 • Fax: 457-0110 • Web: www.emacinc.com

ad index

Advertiser	URL	Page
ARM/KEIL	KEIL.COM	10
ARROW	WWW.ARROWAC.COM/CREE	15
EXPRESS LOGIC	WWW.RTOS.COM	C4
FUTURE ELECTRONICS	WWW.FUTUREELECTRONICS.COM/VIDEO TOUR	21
GEIST TECHNOLOGY	GEISTTEK.COM	37
GREEN HILLS SOFTWARE INC	WWW.GHS.COM	1
HANSOFT AB	WWW.HANSOFT.SE	27
MENTOR GRAPHICS	WWW.MENTOR.COM/EMBEDDED	CV3
MICRO DIGITAL	WWW.SMXRTOS.COM/USB	16
MICROCHIP	WWW.MICROCHIP.COM/PIC KIT3	2
MOUSER ELECTRONICS	WWW.MOUSER.COM	4/6
NATIONAL INSTRUMENTS	NI.COM/EMBEDDED	8
NUMONYX	WWW.NUMONYX.COM/EMBEDDED	13
SMART BEAR SOFTWARE	WWW.CODEREVIEWBOOK.COM	37
TECH TOOLS	WWW.TECH-TOOLS.COM	37
TECHNOLOGIC SYSTEMS	WWW.EMBEDDEDARM.COM	14
TEKTRONIX	WWW.TEKTRONIX.COM/THESOLUTION	18

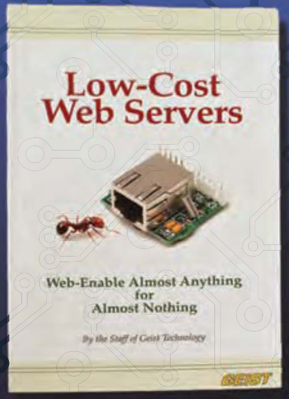
Tired of traveling to check on your remote sites?

Need to add web access to your products?

Let us Web Enable your gear.

GEIST Technology

- Get to market in weeks not months
- Let us do the design work for you
- We are experts in control and data acquisition
- Set alarms, get alerts via SNMP and email
- We have field proven hardware and software


Low-Cost Web Servers
Web-Enable Almost Anything for Almost Nothing
By the Staff of Geist Technology
GEIST

From Web Enabling power control products to monitoring climate conditions at remote sites, we get products online in record time.

Visit GeistTek.com to learn more

Free Book!

Contact us for this free book at (512) 331-8676 or Book@GeistTek.com

By Jack G. Ganssle

Programming quotations

O death, where is thy sting?
O grave, where is thy victory?

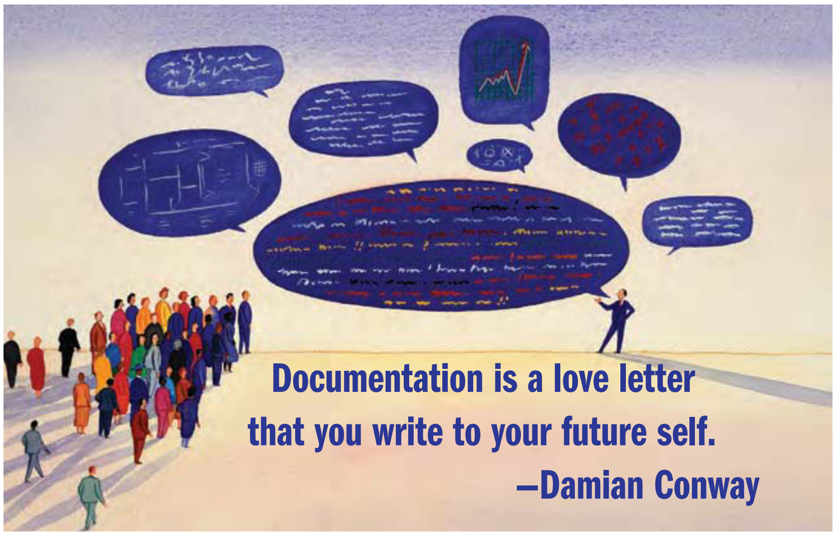
Those words ring with Shakespearian power. *Hamlet*? *King Lear*? Actually, the quote comes from the *New Testament*.

I image the Bible is the most-quoted book of all time. My parents would paraphrase a parable or cite a verse to make a point or correct our behavior. The nuns at St. Camillus did, too, and a sure way to curry favor with them was to quote chapter and verse. Even the most secular couples often use 1 Corinthians 13 in their weddings.

The world abounds in aphorisms that convey wisdom to the young, although that advice is usually ignored. Many aphorisms are by unknown authors: "A stitch in time saves nine" (although anyone who has repaired a sail knows that one stitch can actually save 9,000). "People who live in glass houses shouldn't throw stones." And many witty people have contributed their own, like this gem from Mae West: "Lead me not into temptation; I can find the way myself."

Pithy quotes and aphorisms can entertain and instruct. Often they do both.

Engineers can turn a clever phrase as well as Mae West. I've collected quips that relate, sometimes unintentionally, to development for many years. Here's some of the best:


Documentation is a love letter that you write to your future self.
–Damian Conway

Testing by itself does not improve software quality. Test results are an indicator of quality, but in and of themselves, they don't improve it. Trying to improve software quality by increasing the amount of testing is like trying to lose weight by weighing yourself more often. What you eat before you step onto the scale determines how much you will weigh, and the software development techniques you use determine how many errors testing will find. If you want to lose weight, don't buy a new scale; change your diet. If you want to improve your software, don't test more; develop better.

—Steve McConnell

Programming can be fun, so can cryptography; however, they should not be combined.

—Charles Kreitzberg and Ben Shneiderman

The sooner you start to code, the longer the program will take.

—Roy Carls

Our developers never release code. Rather, it tends to escape, pillaging the countryside all around.

—The Enlightenment Project

Perl is the crystal meth of programming: it's so incredibly useful when you need to do a large amount of work in a small amount of time that you tend to overlook the fact that it's basically precipitating the implosion of your vital organs.

—Dan Martinez

Programmers are the tools for converting caffeine into code.

—Unknown

If you lie to the compiler, it will get its revenge.

—Henry Spencer

There are only two industries that refer to their customers as users.

—Edward Tufte

Einstein argued that there must be simplified explanations of nature, because

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. Contact him at jack@ganssle.com.

God is not capricious or arbitrary. No such faith comforts the software engineer.
—Fred Brooks, Jr.

To iterate is human, to recurse divine.
—L. Peter Deutsch

Good judgment comes from experience, and experience comes from bad judgment.
—Fred Brooks

There are two ways to write error-free programs; only the third works.
—Alan J. Perlis

When Leo Tolstoy wrote *Anna Karenina*, he could have been thinking about cubicles: "there are no conditions of life to which a man cannot get accustomed, especially if he sees them accepted by everyone around him."
—Leo Tolstoy

The real value of tests is not that they detect bugs in the code but that they detect inadequacies in the methods, concentration, and skills of those who design and produce the code.
—C.A.R. Hoare

The most important single aspect of software development is to be clear about what you are trying to build.
—Bjarne Stroustrup

Most of you are familiar with the virtues of a programmer. There are three, of course: laziness, impatience, and hubris.
—Larry Wall

I did say something along the lines of C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do, it blows your whole leg off.
—Bjarne Stroustrup

There are two ways of constructing a software design. One way is to make it so simple that there are obviously no deficiencies. And the other way is to make it so complicated that there are no obvious deficiencies.
—C.A.R. Hoare

Any fool can use a computer. Many do.
—Ted Nelson

Trying to outsmart a compiler defeats much of the purpose of using one
—Brian W. Kernighan and P. J. Plauger

UNIX is simple. It just takes a genius to understand its simplicity.
—Dennis Ritchie

Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.

Putt's Law: Technology is dominated by two types of people—those who understand what they do not manage and those who manage what they do not understand.

An organization that treats its programmers as morons will soon have programmers that are willing and able to act like morons only.
—Bjarne Stroustrup

Theoretically, software is the only component that can be perfect, and this should always be our starting point.
—Jesse Poore

If the code and the comments disagree, both are probably wrong.
—Norm Schryer

Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.
—Damian Conway

Documentation is a love letter that you write to your future self.
—Damian Conway

If you think good architecture is expensive, try bad architecture.
—Brian Foote and Joseph Yoder

Theory is when you know something, but it doesn't work. Practice is when something works, but you don't know why. Programmers combine theory and practice: nothing works and they don't know why.
—Unknown

If the code and the comments disagree, then both are probably wrong.
—Unknown

Those who want really reliable software will discover that they must find means of avoiding the majority of bugs to start with, and as a result, the programming process will become cheaper. If you want more effective programmers, you will discover that they should not waste their time debugging, they should not introduce the bugs to start with.
—Edsger Dijkstra

In theory there is no difference between theory and practice. In practice there is.
—Yogi Berra

For a successful technology, honesty must take precedence over public relations for nature cannot be fooled.
—Richard Feynman

One of the main causes of the fall of the Roman Empire was that, lacking zero, they had no way to indicate successful termination of their C programs.
—Robert Firth

Hofstadter's Law: It always takes longer than you expect, even when you take into account Hofstadter's Law.

PHP is a minor evil perpetrated and created by incompetent amateurs, whereas Perl is a great and insidious evil, perpetrated by skilled but perverted professionals.

—Jon Ribbens

If you want a girlfriend, avoid working in the computer games industry like the plague. If you work seven days a week, 15 hours a day for almost two years, with barely enough time for a pint, you have no time whatsoever for relationships. Plus computer-games makers are regarded as being about as hip and cool as abattoir workers.

—Toby Gard

Your problem is another's solution; your solution will be his problem.

—Unknown

Embedded lines of code are growing 26% annually but developers are increasing by 8%.

—Venture Development Corporation

Productivity can decrease by as much as 25% when workers put in 60+ hour weeks for a prolonged time. And, turnover is nearly three times higher among workers who work extended hours. Absenteeism among companies with extended hours is more than twice the national average.

—Reworded from Circadian Technologies Shiftware Practices 2005 survey

There's a fine line between being on the leading edge and being in the lunatic fringe.

—Frank Armstrong

Two things are infinite: the universe and human stupidity; and I'm not sure about the universe.

—Albert Einstein

Some people, when confronted with a problem, think I know, I'll use regular expressions. Now they have two problems.

—Jamie Zawinski

Debugging is like alien abduction. Large blocks of time disappear, for which you have no explanation.

The most amazing achievement of the computer software industry is its continuing cancellation of the steady and staggering gains made by the computer hardware industry.

—Henry Petroski

One test is worth a thousand opinions.

—Unknown

If the lessons of history teach us anything it is that nobody learns the lessons that history teaches us.

—Unknown

The trouble with the world is that the stupid are cocksure and the intelligent are full of doubt.

—Bertrand Russell

Debugging is like alien abduction. Large blocks of time disappear, for which you have no explanation.

—Unknown

Most software today is very much like an Egyptian pyramid with millions of bricks piled on top of each other, with no structural integrity, but just done by brute force and thousands of slaves.

—Alan Kay

If builders built buildings the way programmers wrote programs, then the first woodpecker that came along would destroy civilization.

—Gerald Weinberg.

Ugly programs are like ugly suspension bridges: they're much more liable to collapse than pretty ones, because the way humans (especially engineer-humans) perceive beauty is intimately related to our ability to process and understand complexity. A language that makes it hard to write elegant code makes it hard to write good code.

—Eric S. Raymond

Let us change our traditional attitude to the construction of programs. Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.

—Donald Knuth

The most unsuccessful three years in the education of cost estimators appears to be fifth-grade arithmetic.

—Norman R. Augustine

No engineer looks at a television remote control without wondering what it would take to turn it into a stun gun. No engineer can take a shower without wondering if some sort of Teflon coating would make showering unnecessary. To the engineer, the world is a toy box full of suboptimized and feature-poor toys.

—Scott Adams

I love deadlines. I like the whooshing sound they make as they fly by.

—Douglas Adams

In handling resources, strive to avoid disaster rather than to attain an optimum.

—Butler Lampson

There are various reasons that software tends to be unwieldy, but a primary one is what I like to call "brittleness". Software breaks before it bends, so it demands perfection in a universe that prefers statistics.

—Jaron Lanier

People tend to overestimate what can be done in one year and to underestimate what can be done in five or ten years.

—Joseph Licklider

Code generation, like drinking alcohol, is good in moderation.

—Alex Lowe

Do you have favorites? Post them on embedded.com. ■

LOWER YOUR BOM

NOT YOUR EXPECTATIONS

MEET THE OS DEPLOYED IN 1.4 BILLION DEVICES

Embedded Systems Solutions | Easier said than done, right? Well here's an embedded operating system that delivers blistering fast real-time performance at a fraction of the cost. The small footprint of the Nucleus® OS minimizes memory and power requirements, which keeps hardware costs down and performance up. Nucleus must be doing something right. To date, Nucleus has been deployed in over 1.4 billion mobile phones worldwide. To make sure your design expectations are reached, go to **www.mentor.com/embedded** or call us at **800.547.3000.**

NUCLEUS

Mentor Graphics®

EMBEDDED EFFICIENCY

BUILD it [Reliably]

With Express Logic's award-winning BenchX® IDE or use tools from over 20 commercial offerings including those from ARM, Freescale, Green Hills, IAR, Microchip, MIPS, Renesas, and Wind River.

RUN it [Fast]

With Express Logic's small, fast, royalty-free and industry leading ThreadX® RTOS, NetX™ TCP/IP stack, FileX® FAT file system, and USBX™ USB stack.

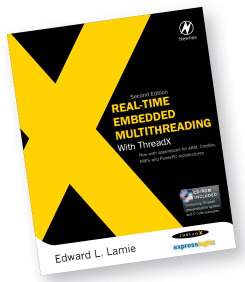
ANALYZE it [Easily]

With Express Logic's graphical TraceX® event analysis tool, and new StackX™ stack usage analysis tool. See exactly what is happening in your system, which is essential for both debugging and optimization.

SHIP it [Confidently]

No matter what "it" is you're developing, Express Logic's solutions will help you build it, analyze it, run it, and ship it better and in less time. Join the success of over 600,000,000 deployed products using Express Logic's ThreadX!

BENCHX THREADX TRACEX STACKX


Newnes

Second Edition

REAL-TIME EMBEDDED MULTITHREADING

With ThreadX

Now with appendices for ARM, Coldfire, MIPS and PowerPC architectures

CD-ROM INCLUDED

Containing ThreadX demonstration system and C code examples

THREADX

expresslogic

Edward L. Lamie

expresslogic

For a free evaluation copy, visit www.rtos.com • 1-888-THREADX

ThreadX, BenchX, TraceX and FileX are a registered trademarks of Express Logic, Inc. All other trademarks are the property of their respective owners.