Embedded COMPUTING DESIGN

www.embedded-computing.com
APRIL 2008 VOLUME 6 NUMBER 3

IDE tools of the trade

-40º to +70ºC Fanless 1GHz Industrial SBC. List Price $595

WinSystems' EBC-855 is rugged, reliable, and ready for harsh demanding applications.

Windows® and Linux Quick Start Kits available.

If you need a powerful PC with fanless operation, choose WinSystems' EBC-855.

This extended temperature, high-performance, x86-based SBC operates Linux, Windows XP embedded, and other RTOSes to provide a seamless development platform. It also supports popular video and wired/wireless network standards.

EBC-855 features include:

- Intel® 1 GHz ZCD CPU or 1.8 GHz Pentium® M
- Intel® Extreme Graphics 2 technology supports CRT & LVDS flat panels simultaneously with dual independent displays
- Custom splash screen on start up
- 10/100 Mbps Intel® Ethernet controller
- 802.11a/b/g wireless supported
- 4 serial COM ports and 4 USB 2.0 ports
- 48 bi-directional TTL digital I/O lines
- Bi-directional LPT port
- Two EIDE ports (UDMA100) for hard disk
- 3.5-in. floppy disk drive and CompactFlash supported
- PC/104 and PC/104-*Plus* connectors
- Onboard AT keyboard and FDC controller
- AC97 six channel 5.1 surround sound
- +5 volt only operation
- EBX-size: 5.75" x 8.0" (146mm x 203mm)
- Industrial temperature operation
- Long-term product availability
- Off-the-shelf delivery

Contact us for additional information or OEM pricing. Our factory application engineers look forward to working with you.

Call 817-274-7553 or
Visit www.winsystems.com/EBC-855
Ask about our 30-day product evaluation

WinSystems®
An Employee-Owned Company

715 Stadium Drive • Arlington, Texas 76011
Phone 817-274-7553 • FAX 817-548-1358
E-mail: info@winsystems.com

Embedded COMPUTING DESIGN

Volume 6 • Number 3   www.embedded-computing.com

Columns

UNIT UNDER TEST

Cover/Web Resources/Events

On the cover

Advanced IDE tools can help developers improve multimedia data processing, COTS software security and reliability, and file system management, as discussed in our Software Feature section starting on page 30.

E-casts

www.embedded-computing.com/ecast

RTOS Trends and Development Environments in Mil Applications
April 24, 2 p.m. EDT

E-letter

www.embedded-computing.com/eletter

Modern test systems strike it rich with data management strategies
By Joe Spinozzi, Cyth Systems

Web Resources

Subscribe to the magazine or E-letter at:
www.opensystems-publishing.com/subscriptions

Industry news:

Read: www.embedded-computing.com/news
Submit: www.opensystems-publishing.com/news/submit

Submit new products at:

www.opensystems-publishing.com/np

Events

ESC Silicon Valley
April 14-18 • San Jose, CA
www.cmpegevents.com/web/esv/home

Systems & Software Technology Conference
April 29-May 2 • Las Vegas, NV
www.sstc-online.org

Features

Special Feature

Hardware

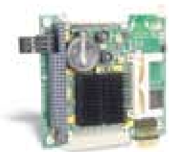
Software

We've Slashed Slow Boot-Up Times

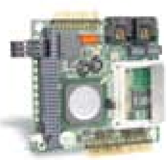
We don't pussyfoot around. Your embedded application needs to come up and execute quickly. Micro/sys CPU boards are optimized to cut through the boot code, which slashes the time from start-up to application readiness for maximum performance over a wide range of hardware platforms.

When you use Micro/sys boards, you gain the value of more than 30 years experience in embedded applications. Our sales team will help you match your boot time requirements with the best board, BIOS, and operating system combination. If you have a tough spec, give us a call.... We welcome the challenge.

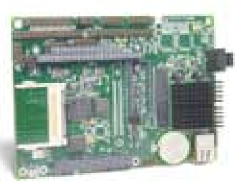
MCB58: Compact microcontroller with CAN, 32 DIO, 4PWM D/A, Temp. Sensor, RTC, 16K Serial EEPROM, 60K Flash, and 4K RAM. *Boots in 160 microseconds.*



SBC1586: Headless, Low-Power PC/104 Pentium with Compact Flash, 4 COM ports, USB, and Ethernet. *Boots in under 4 seconds.*



SBC1625: Low-Power ARM processor with Dual Ethernet, 4 COM ports and 24 Digital I/O on PC/104. *Boots in under 14 seconds to Linux Prompt.*



SBC4495: 486/586 EPIC CPU with GPS, Ethernet, 24 bit I/O, 14 bit A/D and D/A. *Boots in under 4 seconds.*

MICRO/SYS

3730 Park Place, Montrose, CA 91020
Voice (818) 244-4600 Fax (818) 244-4246
info@embeddedsys.com

www.embeddedsys.com

Embedded COMPUTING DESIGN

OpenSystems Publishing™

Embedded and Test & Analysis Group

- Embedded Computing Design
- Embedded Computing Design E-letter
- Embedded Computing Design Resource Guide
- Industrial Embedded Systems
- Industrial Embedded Systems E-letter
- Industrial Embedded Systems Resource Guide
- Unit Under Test
- Unit Under Test E-letter

Editorial Director Jerry Gipper
jgipper@opensystems-publishing.com

Contributing Editor Don Dingee

Senior Associate Editor Jennifer Hesse
jhesse@opensystems-publishing.com

Assistant Editor Robin DiPerna

European Representative Hermann Strass
hstrass@opensystems-publishing.com

Special Projects Editor Bob Stasonis

Art Director David Diomede
Senior Designer Joann Toth
Graphic Coordinator Sandy Dionisio

Senior Web Developer Konrad Witte
Web Content Specialist Matt Avella

Circulation/Office Manager Phyllis Thompson
subscriptions@opensystems-publishing.com

OpenSystems Publishing

Editorial/Production office:
16872 E. Avenue of the Fountains, Ste 203, Fountain Hills, AZ 85268
Tel: 480-967-5581 ■ Fax: 480-837-6466
Website: www.opensystems-publishing.com

Publishers John Black, Michael Hopper, Wayne Kristoff

Vice President Editorial Rosemary Kristoff

Communications Group

Editorial Director Joe Pavlat
Managing Editor Anne Fisher
Senior Editor (columns) Terri Thorson
Technology Editor Curt Schwaderer
European Representative Hermann Strass

Military & Aerospace Group

Group Editorial Director Chris Ciufo
Associate Editor Sharon Schnakenburg
Senior Editor (columns) Terri Thorson
Senior Associate Editor Jennifer Hesse
European Representative Hermann Strass

ISSN: Print 1542-6408, Online 1542-6459

Embedded Computing Design is published 8 times a year by OpenSystems Publishing LLC., 30233 Jefferson Ave., St. Clair Shores, MI 48082.

Subscriptions are free to persons interested in the design or promotion of embedded computing systems. For others inside the US and Canada, subscriptions are $56/year. For 1st class delivery outside the US and Canada, subscriptions are $80/year (advance payment in US funds required).

Canada: Publication agreement number 40048627
Return address: WDS, Station A, PO Box 54, Windsor, ON N9A 615

POSTMASTER: Send address changes to ***Embedded Computing Design***
16872 E. Avenue of the Fountains, Ste 203, Fountain Hills, AZ 85268

Editor's Foreword

Jerry Gipper

The depths of the Grand Canyon

I recently completed a backpacking trip to the Grand Canyon with my Scout troop. Anyone who has ever ventured into the depths of the Grand Canyon understands how challenging the effort can be, especially if you are not properly prepared. In reflecting on the excursion, I was intrigued by how similar it was to the challenges of traversing the business world.

To plan for our hike, we did a lot of research and tapped into the wealth of experience within the troop. We studied trails for their vistas and challenges (our rewards), just like how companies conduct market research before entering into a market. We planned menus and selected gear. Experience taught us to schedule the trip during the cooler month of February to avoid the strenuous heat of summer. We also knew that the trails at this time of year could be icy; it had snowed quite a bit earlier this year, so we expected to encounter some snow at the beginning of the descent. This meant that we needed to carry ice walkers to use on our boots during the frosty sections of the trail. Once the entire plan was in place, we were ready to execute our strategy.

Off we went over the edge, gradually descending into the Grand Canyon. We immediately ran into snow that was reasonably easy to hike over, though the trail was steep and slippery. Some members of the crew were loaded too heavily, and the excess weight of their gear caused them to break through the snow with almost every step. They struggled to advance down the trail, much like a company that becomes too ambitious and spends extra energy and resources to advance in the market.

Everyone made it to the top at their own pace. We eventually returned safely from our trek, and its impact affected my crew in different ways. For some, it was an interesting and exciting challenge, one they look forward to tackling again. For others, it taught them a lesson about the importance of planning and preparing, which will make future treks easier for them to navigate. And some will never journey this way again; the simple fact that they survived was inspiration for them to seek other ventures perhaps more suitable for their skill sets.

In a similar manner, the upcoming months will present economic challenges to companies in the embedded computing industry. Some will fail. Others will struggle to make it out of the canyon. Those that have done their research, invested in the necessary preparations, and made sure they're in sound shape will emerge stronger and ready to take on the next challenge.

Feel free to share your comments via e-mail or visit our blog at www.embedded-computing.com to add your comments.

Jerry Gipper
Editorial Director
jgipper@opensystems-publishing.com

Technology Passport

Hermann Strass

Armed for space services

Reaching out into space

Dutch Space, a subsidiary of EADS in The Netherlands, has finished building the European Robotic Arm (ERA) for the European Space Agency (ESA). ERA, shown in Figure 1, courtesy of ESA/D.Ducros, will be attached to the Russian part of the International Space Station (ISS), which already implements one robotic arm, the Canadarm2. With its different types of base points and payload mounting units, Canadarm2 cannot be used on the Russian module. The ERA, however, is designed to assemble and service the Russian segment of the ISS.

ERA consists of two approximately 5 m-long symmetrical arm sections made of carbon fiber, an Embedded Control Computer (ECC), and other components. With a total length of 11 m (36 ft.), ERA can reach out 10 m (33 ft.) into space and can position loads of up to 8,000 kg (17,660 lbs.) with a precision of 3 mm (0.1"). Both endpoints (end effectors) can be used as hands or feet. The robotic servicing device will handle the following tasks:

- Integrate the ISS
- Manipulate larger building blocks
- Exchange small and large replaceable units
- Install, replace, and deploy solar arrays
- Inspect the space station's surfaces
- Control cosmonauts' extravehicular activities, such as space walks

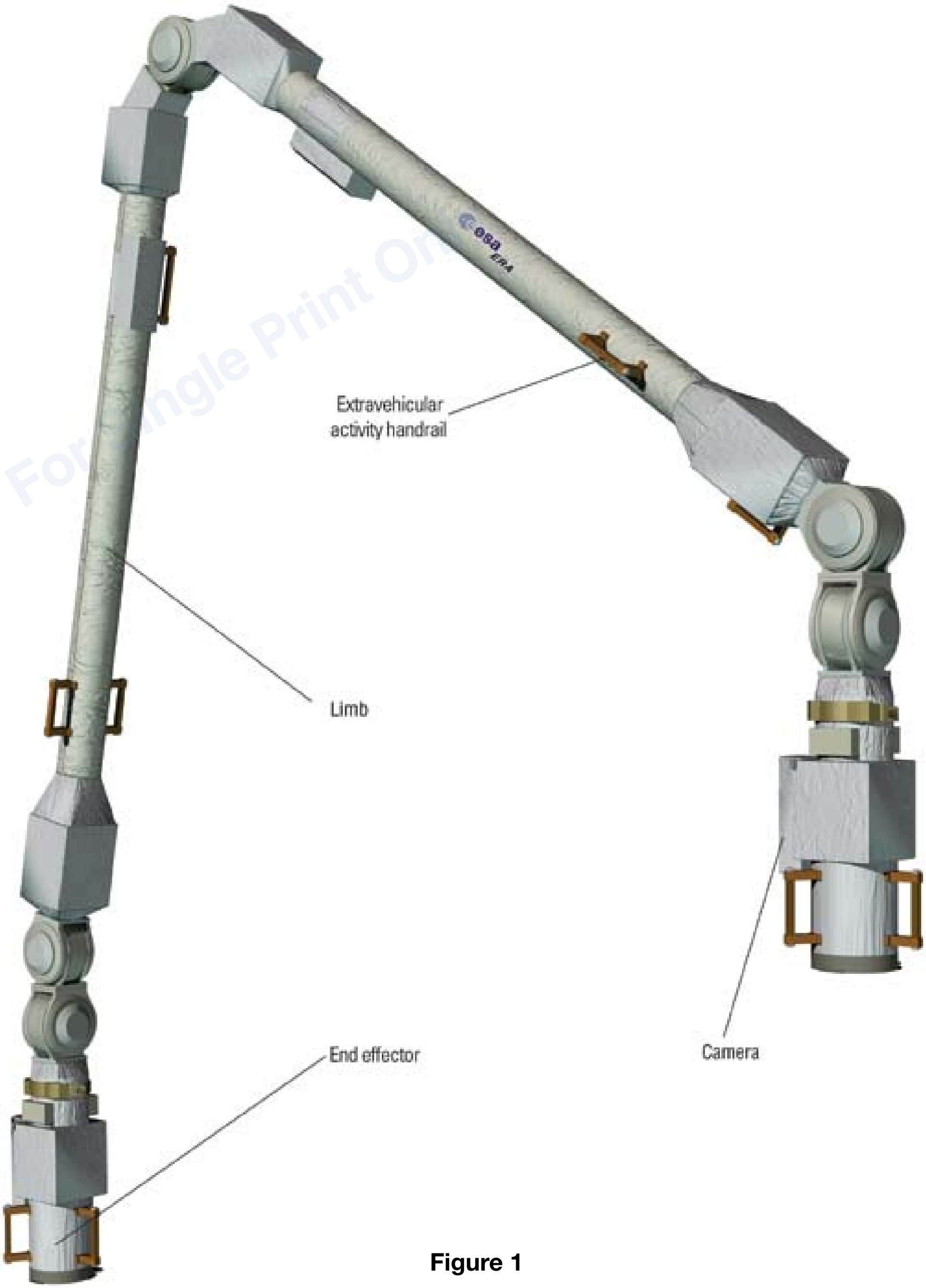


Figure 1

Like a cherry picker crane, ERA will transport cosmonauts to the position where they work or from one external location to another, saving time and effort during space walks. ERA will use infra-red cameras to inspect the space station's external surfaces.

The control computer ECC embedded in the ERA is based on the ERC32 chip, a radiation-tolerant 32-bit RISC processor developed by Temic (now Atmel) for space applications. The ERC32 implementation follows SPARC V7 specifications.

ERA can perform automated, preprogrammed maneuvers via interactive operation control from the ISS and a ground station. Control system software from Terma, Denmark, was used to analyze, design, develop, and validate the onboard control software. This software monitors and controls tasks such as replacing payloads or inspecting and repairing the space station's external surface, including failure detection, isolation, and recovery. Figure 2, courtesy of EADS/Dutch Space, shows the ERA control panel, with switch and button labels displayed in Russian and English.

The software executes on an autonomous computer located on the arm that communicates with sensors and actuators located in the joints, as well as the robotic arm's basic end effectors and cameras. Within the ERA onboard software, a layer of bridging software facilitates communication with the station's mission computer. The software was developed in Ada with support from Technospazio. This language is well matched to the design approach with Hierarchical Object-Oriented Design.

The Russian space agency Roskosmos has announced that its Multipurpose Laboratory Module, slated to launch from the Baikonur Cosmodrome in 2009, will carry ERA up to the ISS.

European awards

Germany's *elektro Automation* magazine recently presented the results of the product contest held during the SPS/IPC/Drives automation exhibition and conference in Nuernberg last November. A panel of industry experts presented a list of 49 products vying for the Automation Award.

The winner – the Handheld Mobile Computer MC9090ex from BARTEC GmbH, Germany – features WLAN/Bluetooth connectivity, long-lasting lithium-ion batteries, and application in explosive environments. Turck's metal sensor and analyzer, Euchner's electronic key system, and Kontron's service-free ThinkIO computer for C-clamp mounting also won awards.

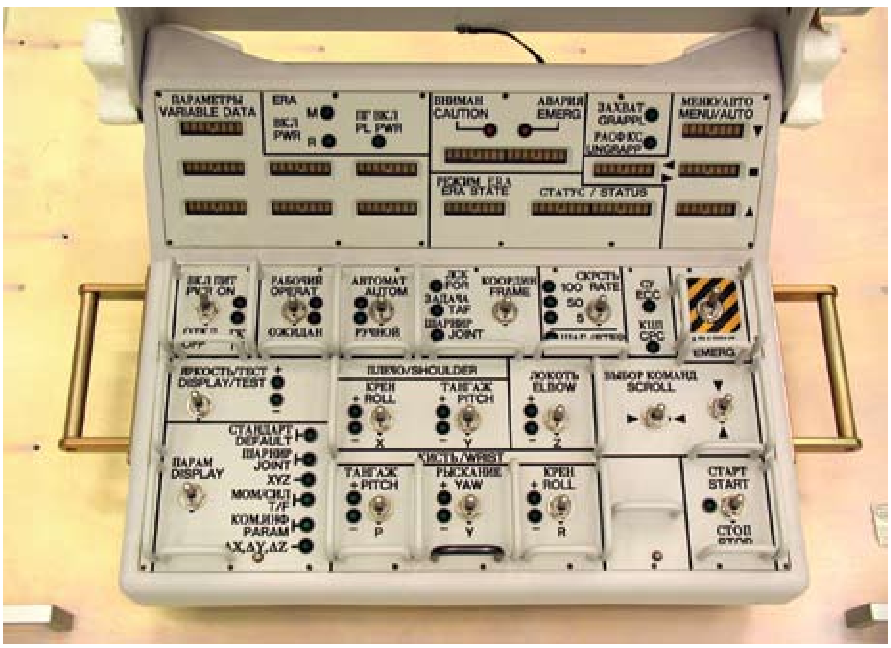


Figure 2

Consortia Connection

Core API, CompactFlash, and wireless USB expand functionality

Khronos Group

The Khronos Group (www.khronos.org) is focused on creating open standards, such as OpenGL, glFX, OpenKODE, OpenVG, OpenGL ES, OpenMAX, OpenSL ES, OpenWF, OpenML, and COLLADA to enable dynamic media authoring and acceleration on a wide variety of platforms and devices.

On February 11, the Khronos Group announced the public release of the OpenKODE 1.0 specification, a royalty-free, cross-platform open standard that bundles a set of native APIs to provide increased source portability for rich media and graphics applications. Khronos also announced a collaboration with the FreeKODE Project to create an open source version of OpenKODE.

A small and light abstraction layer, the new OpenKODE Core API will be familiar to POSIX and C programmers for accessing operating system resources while minimizing source changes when porting applications between Linux, Rex/Brew, Symbian, Windows Mobile, and Real-Time Operating System (RTOS)-based platforms. OpenKODE Core provides advanced functionality, such as multithreading under an event-driven architecture, while providing real-world portability to a variety of mobile platforms. An OpenKODE Core library is typically under 100 KB.

OpenKODE

CompactFlash Association

The CompactFlash Association (CFA, www.compactflash.org) promotes CompactFlash as a worldwide, ultra-small, removable storage and I/O standard for capturing and transporting digital data, audio, and images and performing I/O functionality in modems, Ethernet, serial, Bluetooth wireless, laser scanning, and other technologies.

Last July, a working group was established to develop the CFast specification for a CompactFlash card with a Serial ATA (SATA) interface. Earlier this year, CFA revealed results of this work. Canon executive and CFA chairman of the board Shigeto Kanda remarks, "The development of a CompactFlash card with a SATA interface will maintain the dominance of CompactFlash in the nonconsumer (embedded systems, SBCs, data recorders, and so on) markets as well as promote its use in other applications, such as digital single-lens reflex cameras and professional video cameras."

The current Parallel ATA (PATA) interface provides an interface data rate up to 133 MBps. The SATA interface will provide interface data rates up to 3 Gbps as well as compatibility with SATA disk drive interfaces.

USB Implementers Forum

The USB Implementers Forum, Inc. (www.usb.org) provides a support organization for advancing and adopting Universal Serial Bus technology. The forum facilitates high-quality, compatible USB peripheral development while promoting the benefits of USB and the quality of products that pass compliance testing.

Among the new features for Wireless USB is a new method for first-time device association – the process to securely connect hosts and devices. Wireless USB 1.1 will support Near Field Communication (NFC) capabilities, a proximity-based approach that allows users to introduce devices to their PCs through touch-and-go action. The Wireless USB 1.1 specification will include updates to enhance power efficiency and add Ultra-Wideband (UWB) upper band support for frequencies at 6 GHz and above.

Based on the WiMedia Alliance UWB Common Radio Platform, wireless USB combines the speed and security of wired Hi-Speed USB with wireless technology's ease of use. It is backward compatible with wired USB, allows users to connect up to 127 devices, and delivers a bandwidth of up to 480 Mbps at 3 meters and 110 Mbps at 10 meters.

CERTIFIED WIRELESS USB™

TM & © 2004 USB-IF. All rights reserved.

Ingenuity@work

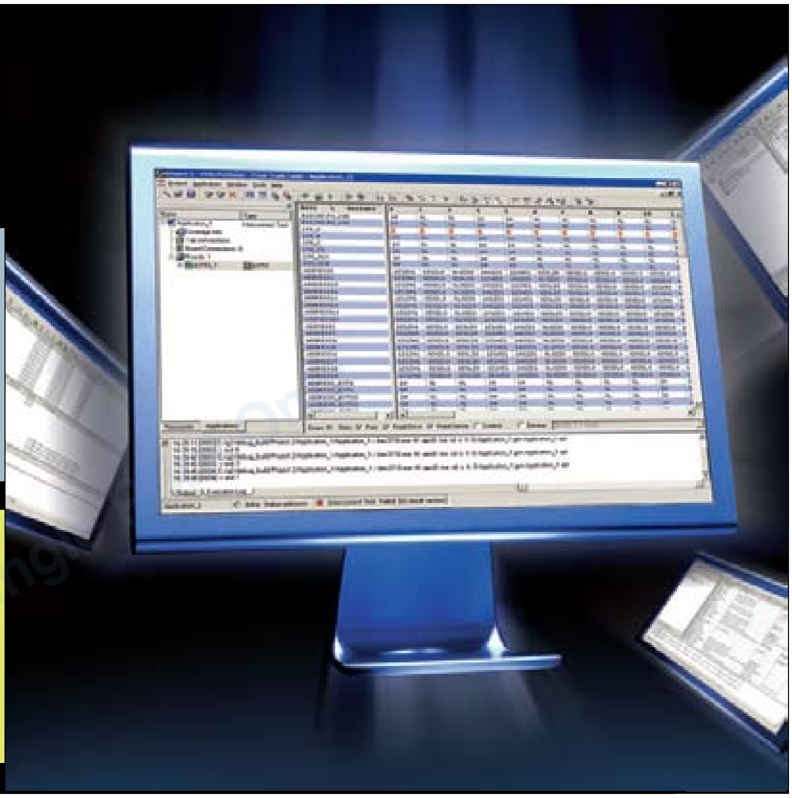
Complex PCB testing

problem

Circuit access problems often arise in testing environments when designers use advanced IC packages such as ball grid arrays. Designers also constantly run into issues dealing with interconnections between memory devices and other non-boundary-scan devices.

solution

Boundary-scan tools enable accurate testing and high-speed in-system programming for densely packed PCBs. Automated, precise fault location detection and coverage analysis reports simplify troubleshooting and allow designs to be optimized before layout.



Quick facts

JTAG Technologies
Founded: 1993
Headquarters: Eindhoven, The Netherlands
URL: www.jtag.com

Boundary-scan like a pro

Testing is a never-ending challenge for any designer. Devices at all levels from components to boards to systems continually evolve and become more complex, making testing even more complicated. Components like processors, I/O chipsets, and Systems-on-Chip (SoCs) have shrunk in size while gaining more functionality and increased performance levels. These types of devices adopted the JTAG boundary-scan standard for testing several years ago, but JTAG remains a difficult tool for designers to fully implement and get the best test coverage possible.

To ensure optimal test coverage, test development for interconnections between memory devices and other non-boundary-scan devices should be automated and fault diagnostics should be prepared automatically. JTAG Technologies accomplishes this with its ProVision JTAG tools. Using ProVision, a designer can quickly prepare tests and in-system programming routines and then examine and manage the details.

Based on the IEEE 1149.1 specification, these boundary-scan tools provide comprehensive coverage and simplify an embedded computing system designer's job by identifying specific nets, components, and pins in the PCB schematic and layout. Automation combined with high levels of control and precision lets engineers maximize their designs and develop their boundary-scan applications quickly, shortening time to market.

JTAG has been around for nearly 20 years and is still evolving. Looking to the future, complex new devices will continually need better internal testing, and system-level products will want to reap the benefits of boundary scanning. New advancements in JTAG will seek to provide System-level JTAG (SJTAG) for multiboard systems and Internal JTAG (IJTAG) for IC-level testing. The IEEE standards working groups have associated test standards already in definition. ECD

AMAX®

Imagination to Innovation

The right manufacturing partner makes all the difference.

AMAX offers comprehensive prototype development, custom manufacturing, and logistics services that deliver robust quality and performance, rapid time to market, and reliable support for your products.

From custom chassis, bezels, and faceplates to your uniquely branded labels and packaging, AMAX provides quality products with a look and feel that reflects your company identity. With AMAX's global logistics and support services, you can deploy your system anywhere in the world with the necessary support to make your product a success.

Utilizing 45nm Quad-Core Intel® Xeon® Processor technology, AMAX systems deliver greater energy efficiency and the multitasking performance necessary to maximize your application with proven reliability.

Voted #1 2006, 2007 CRN Award
Server Leader & Storage Leader

Contact us **(510) 651-8886 / ext. 8800**
or visit us: **www.amax.com.**

Come visit us at:
Embedded System Conference,
Silicon Valley, April 15-17, 2008
McEnery Convention Center. Booth #217

Successfully migrating safety-critical software

By Ole N. Oest

First rule of Software Club: If it's not broke, don't talk about touching it. However, that is just not feasible in many situations, like when code that's working perfectly well has to be migrated for a system-related reason. This becomes a big problem in safety-critical systems, where altering code can trigger a bunch of other expensive, risky activities. So what are designers to do? Here's an explanation of how to gauge what the team is in for and which options should be considered.

Migrating a safety-critical system to a new technology can be a costly, risky process that developers should avoid whenever possible. In some cases, however, migration is desirable for financial or performance reasons or inescapable because of hardware obsolescence and new requirements. Developers facing migration will need to carefully consider the type and extent of system changes to compare the benefits of in-house activity with design service support.

Safety-critical embedded systems deployed in aerospace and defense often have a service life exceeding that of individual system components. The rapid pace of technology evolution creates a high probability that at least one of those components will need to be changed years or even decades before the system itself will be retired. Such hardware changes can, in turn, trigger a need for developers to migrate system software to a new technology to ensure continued serviceability.

A number of system changes can trigger software component migration. For instance, peripherals, communications buses, or protocols may change, forcing code segment migration to new hardware. The target hardware or processor can become obsolete, as was the case with Intel 80860-based systems, compelling the entire system software to migrate to an entirely new platform. New functional requirements or certification standards may arise, forcing the system design to incorporate a Real-Time Operating System (RTOS) where none was needed before. Similarly, the imposition of new standards, new requirements for certification from regulatory agencies such as the FAA, and the need to interoperate with newer systems can generate a need for migrating software to a new platform.

Changes to the development environment also can elicit a need to migrate system software. Obsolescence of the host computer on which the application was developed and maintained, as happened with VAX/VMS hosts, can force system software migration to new development tools when spares for failing hardware become hard to find. Obsolescence of the development tools themselves or loss of expertise in the application tools or language can initiate migration to new tools to ensure that developers can continue supporting installed systems. Similarly, obsolescence of the RTOS can prompt software migration to a new platform.

Even business changes can spur migration. The production royalties associated with an RTOS or another software component can affect system profitability. As margins narrow, developers may choose to migrate system software to eliminate such royalties.

Reducing cost and risk

No matter what triggers the change in hardware or software, migrating system software involves cost and risk. Software migration implies not just changing the software and its attendant risk of introducing errors, but also retesting and possibly recertifying the software. The combined cost of development and test efforts can be considerable, especially for safety-critical systems that must meet strict requirements.

Factors in migration

One key to successful migration – minimizing cost and risk – is thoroughly understanding the impact of migration. Developers need to consider a number of factors, including:

- **Performance:** Will the new processor/RTOS/platform meet the system's real-time deadlines?
- **Resource restrictions:** Will the software fit within the limits of system memory and register availability?
- **RTOS impact:** Changing or adding an RTOS into a once-bare board environment may alter code execution sequence or timing. It also may increase system complexity and alter memory requirements.
- **Word length:** How will changes in word length, say from 16 to 32 bits, affect existing code? Computation algorithms, pointers, counters, overflow/underflow conditions, and execution speed can be influenced by word length change.
- **Tool availability:** Will the host or target platform change also mean a tool set change? The development tools used to create and maintain system software may not be available for a given combination of host system and target processor or RTOS.
- **Data layout:** Compilers vary in the way they map data to registers and memory. Such variations may result in conflicts with implied or expected mappings in the software.
- **Extendibility:** Software migration may require upgrades or enhancements to functionality to meet new requirements. Tools and system resources need to support such enhancements.
- **Traceability:** The ability to trace migrated software back to the original can help reduce test costs by proving that the software has not changed.

The more changes made during migration, the more factors come into play. The lowest-risk migration is to change only one aspect of the system, such as the host development platform. This is feasible if the original software development system and software tools are available on a current host platform such as a PC running Microsoft Windows. Changing only the development host has a minimal impact on the rest of the system and software.

Developers should seek creative ways to keep the number of changes to a minimum. If development tools are not available on the new host platform, for instance, emulation may provide an alternative to switching tool sets. A VAX emulator running on a PC has proven successful in allowing continued tool use, and the binary object code thus generated has typically been identical to the original. The tools, source, and object code did not change, reducing the need for retesting and recertification.

Tool changes require compiler expertise

When tool sets must change, developers face additional challenges. Compilers vary in the way they map the source code to the underlying hardware structure, such as memory addressing and register usage. Unless developers carefully constrain the compiler's behavior, these variations can result in changes to the object code. At best, this triggers a need to retest and possibly recertify the software. At worst, the changes can cause unexpected and potentially flawed system behavior during execution.

Changing tool sets without causing other changes demands that the development team have expertise in compiler behavior – expertise that application-level engineers typically lack. To avoid spending time and effort acquiring the needed skills, development teams can look outside for assistance. Design service organizations usually have experience working with a wide variety of tool sets and can bring that experience to bear in ensuring that a tool change does not trigger software changes.

Designer teams should avoid some changes as much as possible, such as converting the application from a legacy to a current programming language. Instead of converting, teams should utilize a development system for the old language and the new target hardware. This limits the number of concurrent changes and risks to just two: development system and target hardware.

Changing languages involves many possible pitfalls. The generated application will not be identical to the original, requiring costly retesting and recertification. Other factors come into play as well. The generated code will have a different layout and may no longer fit in the available memory; data layout will be different and no longer map correctly to the underlying hardware; and performance and timing aspects will change. The application must be modified at the source code level, which will require training software engineers in the new programming language as well as in the design and inner workings of the application.

Although it might be tempting to migrate to a new language if none of the programmers are trained in the application's programming language, this should be the last resort. Before taking that route,

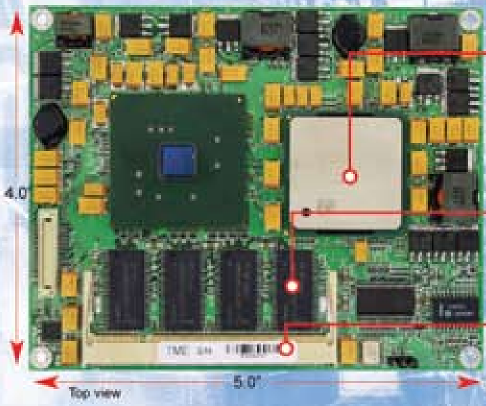
Embedded Computer Solutions For Harsh Environments

-40°C to +85°C Operating Temperature

Five Year Product Availability Guarantee

ECM401

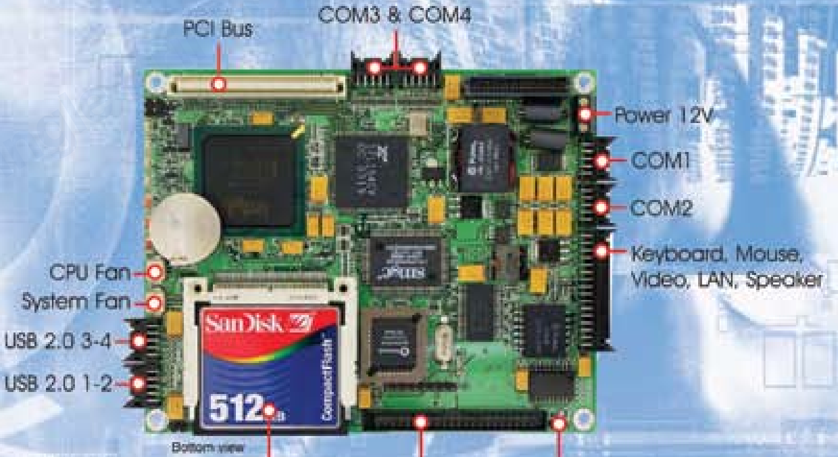
World's Fastest Embedded Computer Module



CPU, up to
2.8 GHz Pentium 4
(Low power 1.6Ghz and 2.0Ghz Pentium 4 also available)

512Mbytes soldered on-board DDR memory expandable to 1.5Gbytes using SODIMM socket

SODIMM socket



Features:

- (6) Serial ports
- (4) USB 2.0 ports
- (1) 10/100Base-T
- Compact Flash Socket
- E-IDE supports 2 devices
- Video I/F (1600X1200, 32 Mb)
- AC'97 2.2 Audio
- Less than 5 seconds boot up time
- Intelligent thermal management with independent microcontroller
- Power requirement: +12V @ 3A (2.0Ghz P4, 256MB)
- Over 200,000 hours MTBF

Over 15 years in business of designing and manufacturing embedded computer products for OEMs and System Integrators

We can design your Embedded Computer Board in 30 days.

Time-to-market is very critical to a company's success.
TME has a proven development and manufacturing process that allows new embedded computer products, using TME's core technologies, to be developed in 30 days.
OEMs and System Integrators can now bring their products to the market without compromising cost and features at the expense of time.

ECM401 Embedded Computer Module provides all functions and features of a high performance computer on a very small Module.

Embedded Computer Designers can easily design an I/O board with proper connectors and form factor that is most suitable for their applications.

The ECM401 supports up to 1.5Gbytes DDR memory, 2.8GHz CPU, with built-in Audio, Video, USB, Network, serial, LPT interfaces and PCI BUS extension capability.

Typical Applications

- ▲ Robotic
- ▲ Medical
- ▲ Avionics
- ▲ e-Kiosks
- ▲ Transportation
- ▲ Military/Aerospace
- ▲ Industrial Automation
- ▲ Inventory management
- ▲ Point Of Sale Terminal
- ▲ Test & Measurement

tme

Toronto MicroElectronics Inc.
5149 Bradco Boulevard, Mississauga, ON., Canada L4W 2A6
Tel: (905) 625 - 3203 ~ **Fax:** (905) 625 - 3717
sales@tme-inc.com ~ www.tme-inc.com

consider training programmers in the old languages. Programmers proficient in relatively complicated current languages such as Java or C++ will not find it insurmountable to learn another language.

Design services offer expert assistance

Another possibility is to engage design services that offer the necessary language expertise. For specialized languages such as Ada and JOVIAL, which target military and avionics systems, design service providers often have extensive experience in the application space as well as the language, including experience with the needs of safety-critical system design. This enables them to quickly develop an in-depth understanding of system software and provide the maintenance and upgrade support the development team requires.

If, in the final analysis, the original language must be scrapped, systems designers can change languages in part using translation tools (such as in Figure 1). No tool, however, can do a complete job, and converted source program readability may be questionable. When possible, development teams should strive to change languages only for sections where it is absolutely necessary.

One way to accomplish this is to use a tool set that supports the old and new target languages and can mix languages. This allows teams to keep intact those sections of original code that are still useable and limit language changes to sections involved in meeting the new requirements.

A key part of such mixed-language tools is the debugger. While many compilers can combine code segments in different languages, most debugger tools handle only one language at a time. This means that developers must invoke several tools simultaneously to view interactions among code segments, and these tools seldom interact in a coordinated fashion or exchange information to help correlate object code to multiple language sources. Tools such as DDC-I's OpenArbor (shown in Figure 2) that allow mixed-language debugging from a single launch can significantly reduce debug time and more readily detect interaction errors.

Whether or not a language change is involved, migrating safety-critical system software is a complex task with many potential pitfalls. Each change in hardware, host, target, tool, and language introduces complications and may force additional changes, leading to escalating consequences. The costs and risks inherent in migration should be avoided as much as possible by maximizing legacy tool and code reuse. When changes are necessary, carefully selecting new tools and strategically using experienced design services can mitigate software migration risks and costs. ECD

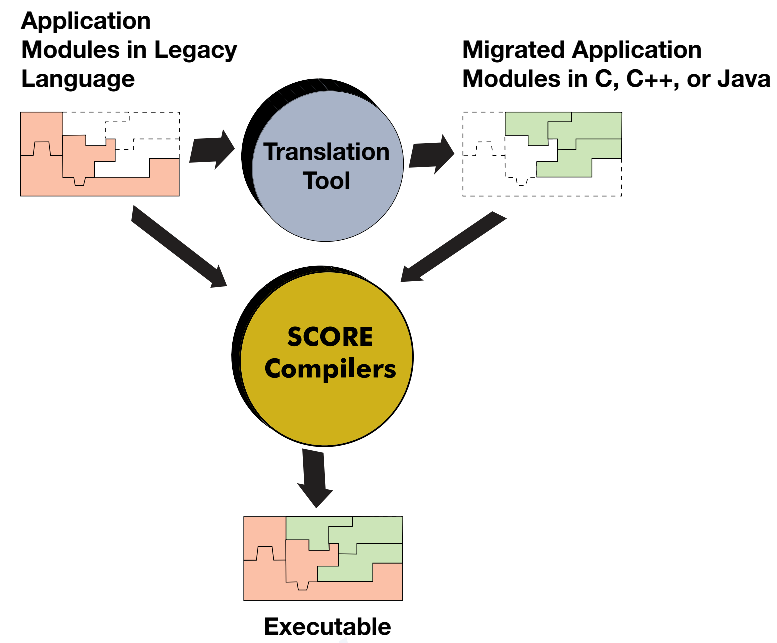


Figure 1

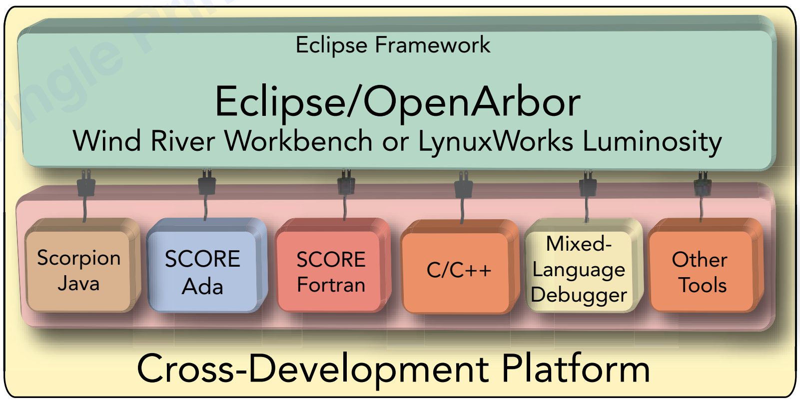


Figure 2

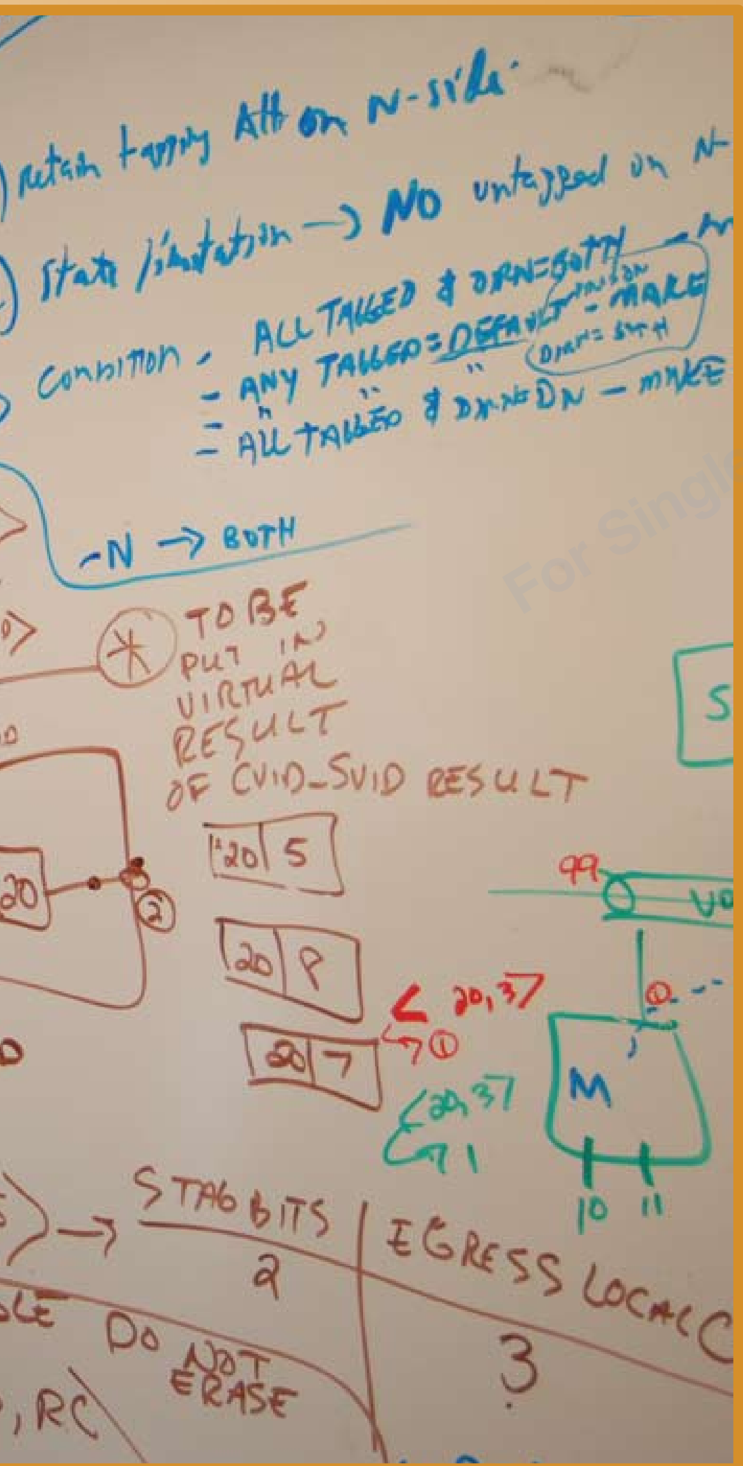
***Ole N. Oest** is Chief Technology Officer and cofounder of DDC-I (Phoenix, Arizona), a leading supplier of software tools and services for safety-critical embedded applications. Ole holds an MS in Electrical Engineering and a PhD in Software Engineering from the Technical University of Denmark, with special interest in programming languages and compiler construction as well as formal specification and program development.*

DDC-I
602-386-4360
ooest@ddci.com
www.ddci.com

Global R&D models take hold

By Stephen Ramponi

While looking at the bottom line and searching for new ways to compete globally, company executives are now realizing that being competitive in the international marketplace not only requires global products, but also a global engineering workforce.



Offshore development evolves

Since 2002, U.S.-based software and hardware product companies have increased offshore outsourcing work to India, China, Russia, Ukraine, and other countries. Total offshore engineering product development has significantly increased in recent years. While cost is still a motivating force, time to market and access to talent and growth are on par with cost as drivers toward R&D globalization. As global R&D models become part of several organizations' engineering strategies, managers are beginning to discover the nuances of offshore operations.

Many companies looking to offshore models have moved beyond the initial hourly rate charged. If not for the cost savings, there would be little impetus to move resources from one location to another. In fact, in today's venture capital-driven economy, this labor arbitrage is what allows new companies to scale quickly while coping with limited cash.

Besides cost, businesses must consider the implications when selecting a partner. Like any other purchase, you get what you pay for in terms of quality and efficiency. This is especially true when recruiting smart engineers to help bring a product forward. Simply employing the cheapest staff may cause productivity and quality to suffer and release dates to slip.

Today, many engineering managers use other methods to evaluate offshore development: Total Cost of Ownership (TCO), quality, and productivity. TCO takes into consideration everything relevant to increased costs including logistics, control, intellectual property, and communication. Quality refers to the final deliverable's perfection as well as documentation, clarity, maintenance, sustainability, scalability, bug fixing, and version control. Productivity requires companies to analyze the effects of time zones, including speed, travel, code integration, team interaction, and company culture.

During the past several years, pure offshore development users have highlighted the major challenges surrounding their engineering work. The most important issues that delay or in some instances doom an offshore development project include:

- High cost of re-creating capital equipment requirements offshore
- Building a tightly integrated team
- Managing projects in different time zones
- Additional energy required to manage an offshore operation
- Logistical issues and time it takes to get work done
- Employee resentment of reducing local staff
- Increasing dependence on offshore resources

Overcoming offshore challenges

Companies encounter these and other difficulties while implementing and managing offshore projects. Many struggle when first starting to work with offshore teams. The problem is common, but the diagnosis is often wrong. The following discussion provides advice on how to avoid the usual pitfalls.

Don't throw it over the wall

The worst thing a company can do is treat its offshore team as a black box, sending them specs and waiting for the work to be completed. This is often where the wheels fall off the project. Instead of a one-way client/vendor relationship, businesses should create an environment where everyone feels like they're part of one seamless team. Both onshore and offshore teams must work on common deliverables and timelines and clearly understand how their performance is being measured.

Apart from internal discussions, firms should keep the remote team informed about the business, not just the local team. This will keep them motivated to focus on meeting release dates and other development milestones. Also, creating the one team dynamic will reduce any offshore versus onshore tensions that can fuel mistrust and lead to poor results.

> "Creating the one team dynamic will reduce any offshore versus onshore tensions…"

Look for a global provider

Many firms say they're global, meaning that they have U.S. salespeople and offshore delivery. But there's value in looking for a provider that can provide engineering resources on-site as well as offshore.

Dual-shore outsourced engineering provides numerous advantages. The initial dual-shore benefit is realized during the analysis and specification stages. Instead of having an engineer halfway around the world nail down the most difficult technical details of a project, local engineers in a dual-shore team can get fully involved in defining and specifying the work to be delivered. By taking over some of the project management responsibilities, the dual-shore solution allows for increased productivity without overtaxing the engineering manager's already full plate.

The science of managing distributed teams

It's true that managing distributed teams is complex. However, having strong processes for transferring knowledge, breaking up tasks, and managing interdependencies between teams can be critical to a global R&D strategy's success. This can be tricky for managers who have never worked in a distributed environment, so companies should seek their partners' advice on how to solve the problem together.

Constant communication is key

One of the most important aspects of a successful outsourcing strategy is having a communication plan that ensures frequent dialog. The goal is to create a collaborative work environment that replicates informal hallway conversations in a more formal way. Being 10 hours ahead of New York and 13 hours ahead of California creates some interesting challenges for teams based in India.

Today's multitude of collaboration and communications technologies can help businesses deal with the time difference and create an approximation of hallway discussions and learning opportunities. Office branches can use blogs, wikis, Skype, and desktop videoconferencing to keep each other apprised of activities, accomplishments, and problems. These actions can help strengthen personal and professional bonds and ensure that fewer messages get lost in translation.

Evaluating project success

The promise of highly skilled, low-cost engineering talent is luring many companies to begin offshoring a large portion of their core and noncore engineering projects. Firms are beginning to implement a more robust method of evaluating project success by looking abroad to meet their engineering workforce needs. **ECD**

***Stephen Ramponi** is VP for Symphony Services' embedded systems business unit in Westford, Massachusetts, where he is responsible for strategic planning, human resources, and recruiting operations. Before Symphony, Stephen was a key member of Viridien Technologies, which provides services within embedded systems and Internet/Web domains. He began his career in the consulting industry at Aerotek Inc., one of the largest, privately held technical staff augmentation companies in the world. Stephen holds a BA from Salve Regina University in Newport, Rhode Island.*

Symphony Services
978-256-1113
sramponi@symphonysv.com
www.symphonyservices.com

Developing an electronic LEGO for embedded systems

By Paul Zawalnyski

Embedded system development can be a bit like reinventing the wheel. Many embedded systems vary only by interface requirements. Given that customers are not willing to pay or compromise the product for functionality they don't want, designers have learned to reuse software. While PC/104 provides off-the-shelf hardware, designers sometimes must look for other form factors when customer requirements call for a battery-powered system that must fit into a matchbox, not a shoebox.

The Compact Computer was the outcome of a customer requirement, although at the time the customer did not know it. This computer is similar to the calculator chip Busicom commissioned Intel to design. Intel called it the 4004, which later became the first microprocessor.

The customer's requirement was simple: Take a real-time data acquisition system for a wind tunnel model and send the data out for analysis either on a single wire or wirelessly. The main problem was that the company only wanted two devices, thus precluding a custom design. The design team could have invested to develop this as a product, but it was not the right time.

Choosing a form factor

So the designers went on a hunt for an appropriate commercially available solution. Many SBCs were available but lacked the right mix of input or output for the application. This search eventually led to PC/104. The team built a suitable PC/104 stack but quickly determined it was too big to fit in the model.

Then the eureka moment arrived when the designers realized that several commercially available boards smaller than PC/104 – PC-Card, CardBus, and CompactFlash, to name a few – ship in the millions of units. However, these boards are normally used in laptop computers, no more than two at a time. Adding the laptop to the equation made the final design too big and too expensive as an embedded system.

At that point, the designers considered how difficult it would be to develop a processor that could talk to several of these cards. It sounded pretty straightforward, as long as all the cards were the same type, not mixing PC-Card with CompactFlash or CardBus. (PC-Card and CardBus have the same form factor, making it hard to determine which is which.) If the designers built a system that could only use PC-Card or CardBus, numerous users would populate the system with the wrong type of cards, creating a support nightmare.

CompactFlash eventually stood out as the optimal form factor for the system. A quick search on the types of CompactFlash cards available produced a list of cards that supported Ethernet, Bluetooth, ZigBee, 802.11, mass storage, GPS, GPRS/GSM, and Analog-to-Digital Converter (ADC)/Digital-to-Analog Converter (DAC). So it appeared that miniature embedded systems could be developed using standard CompactFlash cards just like PC/104 cards, but one-thirtieth the size. The team aimed to build these miniature embedded systems (4 cm x 4 cm x 2 cm) using commonly available low-cost cards with minimal hardware development and nonrecurring engineering costs.

Knowing that they could easily build rapid product prototypes of these systems in hours or days, the designers wondered why no one else had made them. A manager explained that there are two reasons certain products don't exist: it is a stupid idea or nobody has thought of it yet. The designers went with the latter reason as they proceeded with product development.

Software, application considerations

To achieve wide-scale adoption, the CompactFlash computer's software had to be based on a standard Operating System (OS) such as Windows CE or Linux to minimize the learning curve. The goal was to be able to plug in any CompactFlash card and let the system run. The ability to swap Ethernet cards for wireless cards and not make any software modifications was appealing. Since it was possible that the drivers for the CompactFlash cards would need to be modified to support the architecture, the designers chose open source Linux.

At that time, some small Linux systems were on the market, but none as small as 42 mm x 37 mm. After making a few compromises, the team created the first CompactFlash Computer. The rapid prototype with two Ethernet ports and wireless capability is shown in Figure 1. Based on a Freescale MCF5272 microprocessor running at 40 MHz with 8 MB flash and 32 MB SDRAM, the system acted as a CompactFlash host (it controlled other CompactFlash cards) and was originally supplied with a 10-slot motherboard measuring 52 mm x 42 mm. A low-cost 4-slot carrier board was added later. Figure 2 shows the system's carrier board with three CompactFlash expansion slots.

Figure 1

Figure 2

While some companies used the CompactFlash system for rapid prototyping and proof-of-concept engineering, others used it to develop small systems. Projects ranged from wireless-based health-monitoring systems to GPS logging to data storage.

Other designers wanted the ability to plug the system into a CompactFlash card slot in existing devices such as cameras and PDAs. Applications included:

- Secure applications for PDAs
- Accelerators for existing devices to allow new functionality within the original processing budget
- Emulating storage cards in existing equipment and accessing files via wireless or serial communications, cameras, data loggers, and so on
- Additional digital I/O for systems with a CompactFlash slot through a connector on the top; tiny add-on cards could contain ADC or DAC clips with a custom connector

Though these were reasonable applications, the first Compact Computer was designed as a CompactFlash host and could not be plugged into a CompactFlash socket. One of the compromises made in the design was using a small programmable logic device to generate the CompactFlash bus signals, ensuring that the board layout could be achieved in a reasonable number of layer and line widths.

Adding an FPGA

The simple solution was to replace the programmable array logic with an FPGA, which allowed the CompactFlash processor module to manage the bus as a host or slave device and made the bus reconfigurable. Using an FPGA also made it possible to quickly add other features to the module. Multiprocessor systems could be developed with custom communication systems, shared memory, or message passing. In essence, this was an electronic LEGO.

FPGA selection became the critical design decision as the board space had shrunk to 37 mm x 25 mm with the inclusion of a top connector for I/O. An FPGA with a soft-core processor would allow one part to do the work of two. But the 100 MHz performance claimed by vendors with soft-core processor FPGAs quickly decreased to 50 MHz when the FPGA was loaded with other functions. Going from 40 MHz to 50 MHz was no wow factor, especially when similar systems were already at 400 MHz. The new system had to have a processor plus an FPGA.

With a 12 mm x 12 mm package and 500, 600, and 750 MHz clock options, the Analog Devices Blackfin processor gave the new system an easily upgradeable future. It had 0.8 mm solder ball spacing, which simplified layout and manufacturing. Figure 3 shows the latest system with a 500 MHz Blackfin processor.

The designers then turned to the layout problem, iterating through different FPGA, SDRAM, and flash devices until finding a board layout that could be manufactured at reasonable costs. All the computers in the company were running PCB routing programs 24 hours a day, 7 days a week for a long time.

Revolutionary prototype development

Built with commercial hardware at a fraction of the size of existing industrial systems, the Compact Computer worked, meeting the customer's requirements and providing a starting point for future designers. To reap the benefits of this system, designers must not to be constrained by a "nobody else does that" attitude. Instead, designers should change the way prototypes are developed by taking advantage of true plug-and-play embedded systems. After all, you don't win races by following the leader. ECD

Figure 3

Paul Zawalnyski *is the managing director and acting technical director of C Data Solutions, an Exeter, England-based company that was formed in late 2004 specifically to develop the Compact Computer and modular embedded systems. He has 25 years of experience developing hardware and software in the military, medical, and networking industries. Paul earned his 2.1 Degree in Electronics from the Dundee College of Technology in Dundee, Scotland, and his Diploma in Digital Techniques from Heriot-Watt University in Edinburgh, Scotland.*

C Data Solutions
+44 1392 277 202
paulz@cdatas.com
www.cdatas.com

Open standards accelerating next-generation multimedia device delivery

By Fakhir Ansari

Consumers are expecting more and more from their multimedia devices, pressuring application developers to keep up. Fortunately, middleware frameworks with standard components are emerging to help designers. Fakhir introduces the OpenMAX multimedia framework and illustrates how it is changing multimedia device development.

A gradual yet revolutionary change is transforming the way multimedia is used in software applications today. Not long ago, most multimedia vendors had their own implementation. Code interoperability and portability were generally not major requirements. But now, with more powerful hardware and increasing demand from end users, the multimedia domain has expanded in all directions.

This expansion has now reached a level where a single vendor can no longer address all the requirements. Accelerated hardware, codecs, container formats, network streaming, and other very highly specialized subdomains have appeared. This growth has triggered a major shift in the way multimedia services are perceived.

From services to frameworks

To understand this shift, developers must examine traditional multimedia libraries. These libraries normally have a static structure and provide a fixed set of services. The services provided are definitive, such as "playing a WAV file" or "playing an MP3 file." The API itself is vendor-specific, and applications written for one multimedia library are usually not portable to another. Library implementation is kept opaque, limiting options for customization or extension.

To address the growing demands of the expanding multimedia domain, software vendors have shifted their focus to *multimedia frameworks*, illustrated in Figure 1. A framework is a heterogeneous mixture of software from different sources. The key feature of multimedia frameworks is a flexible and extensible architecture that allows the services provided by the framework to evolve with the changing requirements of the industry.

Multimedia framework flexibility is achieved by utilizing the concept of a *component*. Components perform like simple building blocks that fit together

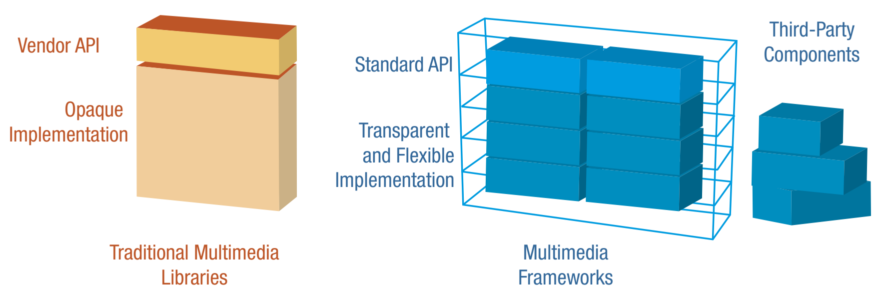


Figure 1

Figure 2

to form more complex systems. A framework API does not provide access to particular services, instead allowing the developer to assemble different components according to design requirements. The framework is independent of what these components actually do and how they do it.

Why has the framework paradigm worked well for multimedia? The answer lies in the nature of multimedia processing. Multimedia processing inherently involves a linear flow of data through different stages. Each stage is well defined and logically independent of other stages. Therefore, a linear arrangement of components in the form of a pipeline naturally suits multimedia. Figure 2 shows a sample pipeline for audio playback. Multimedia data flows in from one end and is processed by different components as it exits the pipeline from the other end.

Multimedia framework advantages

Realizing the power of this concept can be difficult without examples. Frameworks normally contain a rich library of components. Table 1 categorizes four types of components. A framework user will usually select one component from each column of the table and make a pipeline of the resulting four components. It is easy to see that numerous configurations are possible using these sample components. For example, the MP4 demultiplexer, MPEG4 decoder, video scaling, and video output components can be connected together to display video. Adding support for subtitles to this video will be as simple as adding the subtitles component to the pipeline.

One important characteristic of a framework is that each component is loosely coupled with other components and thus is easily replaceable. For example, one may replace a standard video decoder with a hardware-accelerated video decoder. Enhancing existing applications is simplified as the user only adds or replaces existing components with more enhanced versions.

Standardization ensures interoperability

Each component's internal logic is encapsulated in a standard component definition. This standardization and the aforementioned loose coupling provide an excellent platform to ensure interoperability among components written by different software vendors. Several software vendors may contribute to a single framework, and all of their components fit and work together seamlessly. A framework also serves as a software integration tool.

Input	Codec	Transforms	Output
File reader	MPEG4 decoder	Video scaling	Audio output
Network reader	MP3 decoder	Video rotation	Video output
MP4 demultiplexer	MP3 encoder	Video multiplexing	Network writer
MOV file demultiplexer	AAC decoder	Audio volume	File writer
	H.264 decoder	Audio equalizer	
		Echo effect	
		Subtitles	

Table 1

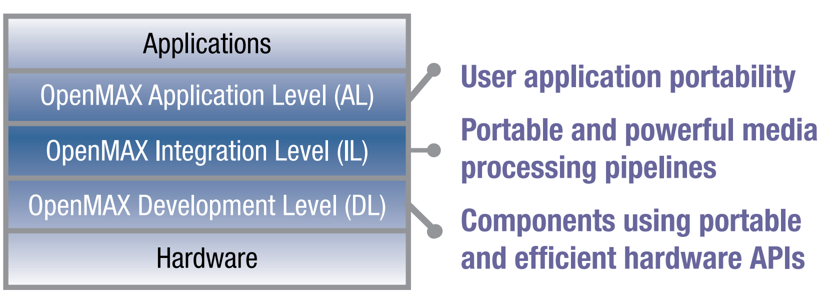


Figure 3

The more popular multimedia frameworks in use today are usually platform dependent. Examples include DirectX for MS Windows and GStreamer for Linux. But standardization has risen a notch higher. Cross-industry groups such as Khronos have standardized the framework definition itself. An open, royalty-free framework definition by a neutral group has encouraged collaboration among software vendors. The multimedia framework defined by Khronos is called OpenMAX (www.khronos.org/openmax/). Although this is a new standard, several companies have already embraced it.

The OpenMAX standard is made up of three levels, as shown in Figure 3. What has been discussed up until now corresponds to the OpenMAX Integration Level (IL), which defines a component-based framework. The other two levels above and below the IL level address equally important aspects of the framework: implementation and usage.

Writing components for multimedia frameworks

The component library is the largest functional area of a multimedia framework and involves the most effort from software and silicon vendors. Vendors usually specialize in certain services; for example, a software vendor may specialize in providing video codecs like MPEG4. This particular codec can become part of a multimedia framework once embedded in the framework's components. Vendors encapsulate services into components to make them standardized and easily pluggable into existing software, opening up opportunities for widespread use of their products.

Another notable characteristic of a multimedia framework is the ease with which it allows third-party services to be integrated into components. Frameworks provide special tools and techniques for this very purpose.

Given that these assistance tools usually vary from one framework to another, this discussion will focus on OpenMAX framework-related features, particularly the Nucleus Multimedia Framework implementation by Mentor Graphics.

Multimedia data processing is extremely time critical. Data must be compressed, decompressed, or converted to other formats in real time. This data processing employs computationally intensive algorithms that must be highly optimized. OpenMAX Development Level (DL) addresses this vital area of optimization, providing an API to a large set of commonly used algorithms related to multimedia processing.

Service providers will not have to worry about implementing and optimizing these algorithms; they simply use the OpenMAX DL API in their software. The actual implementation of these APIs is then provided by another stakeholder in such systems – silicon vendors. A silicon vendor implements all OpenMAX DL-defined algorithms, which are specifically optimized for the vendor's hardware platform. This benefits software vendors by allowing their software to run efficiently on hardware and helps silicon vendors by ensuring that software written for their platforms utilizes the hardware to its full potential.

Framework components carry out many common operations, such as managing buffers, maintaining the component state, and protecting data. Some frameworks simplify the task of component writers by allowing component hierarchies. One generic base component provides all the common functionality and other components that can be derived from this base component, as shown in Figure 4. Using object-oriented design principles, a derived component inherits the properties of a base component, minimizing redundancy and helping component writers focus solely on their specific services.

Because a framework serves as a heterogeneous mixture of software from different sources, a component writer may not always be familiar with another component. This is where the additional debug and development tools provided by the framework come into play. Debug tools are most crucial as they help visualize the multimedia pipeline and locate problems. Figure 5 represents a component pipeline in real time with the Nucleus Multimedia Framework debugger.

Using multimedia frameworks in software applications

Despite the advantages of component-based frameworks, these types of APIs are not readily accepted by application developers, who are accustomed to plain APIs such as "play an MP3 file." Having to create components, connect them together, and then use them – no matter how simple an operation – does not provide a sufficient level of abstraction to justify their use.

The OpenMAX Application Level (AL) seeks to address these concerns, providing an easy-to-use API that hides the mechanisms of the underlying framework. This also makes user applications more portable, as they use an open standard consistent across all hardware platforms instead of relying on a proprietary API.

Recently, some frameworks have moved to an even higher level of abstraction. Instead of providing a programming language API, the developer creates an application by defining it in simple XML. This technique is catching on in

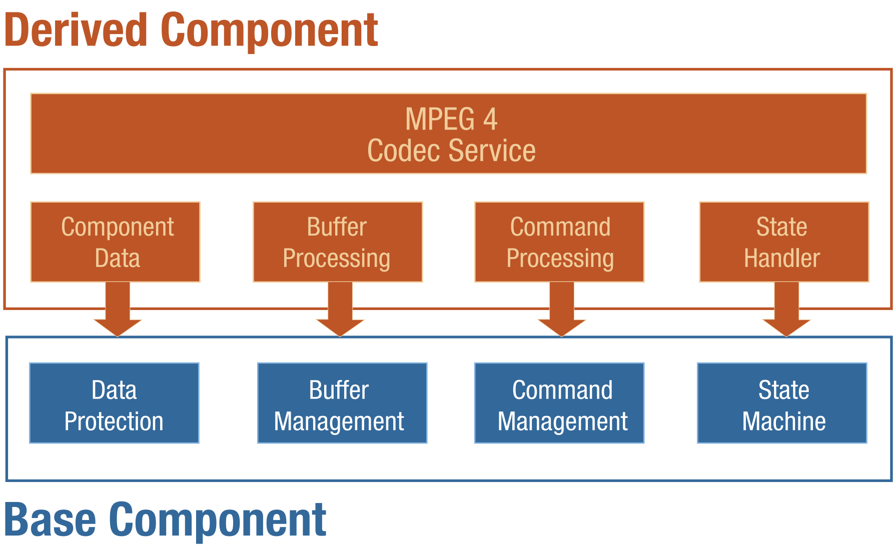


Figure 4

user interface applications. Integrating a multimedia framework at such a high level has allowed multimedia to be used in ways that were not possible until now.

APIs easing integration

The embedded industry is accelerating efforts to establish royalty-free APIs, which enable media authoring and promote adoption across a wide variety of platforms and devices. The Khronos Group is closely involved with these efforts, and its OpenMAX standard for media library portability is gaining serious momentum.

The OpenMAX cross-platform API enables accelerated multimedia components from different software vendors to be developed, integrated, and programmed across multiple operating systems and silicon platforms. With this approach, embedded device integrators can take advantage of library and codec components from any software vendor, as long as they are built on OpenMAX APIs, while realizing the full acceleration potential of new silicon platforms. The result will be devices with the most advanced multimedia capabilities delivered into the hands of consumers at the silicon beat-rate. ECD

***Fakhir Ansari** is a technical lead for multimedia software development at Mentor Graphics Corporation, based in Wilsonville, Oregon. Fakhir has nearly five years of experience in embedded software development, with particular focus on cryptography, networking protocols, and multimedia systems for handheld devices. He is also a software development hobbyist with 10 years of programming experience. Fakhir holds a BS in Computer Science from the National University of Computing and Emerging Sciences in Lahore, Pakistan.*

Mentor Graphics
251-208-3400
fakhir_ansari@mentor.com
www.mentor.com/embedded

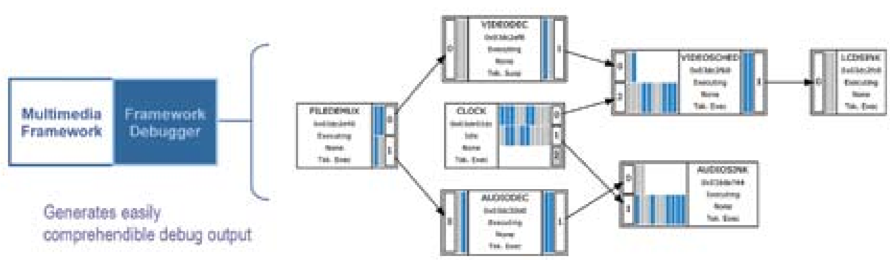


Figure 5

Improving reliability and security by analyzing executables

By David Melski, PhD

Many source code analysis tools available today, including Coverity Prevent, GrammaTech CodeSonar, Klocwork K7, and The MathWorks PolySpace Verifier detect software defects and vulnerabilities. During the past few years, interest has grown in performing similar analyses on executable machine code. Three main factors are driving this interest in direct machine code analysis: the need to control COTS software reliability and security, the technical advantages over source code analysis, and the recent increases in its feasibility and utility, which have been substantiated by breakthroughs in the research community. David explores the advantages of machine code analysis and summarizes the current state of the art.

Taking back control of application reliability and security

Time-to-market and cost demands have increased developers' use of COTS components in embedded software applications. While these components offer advantages, they come at the price of some well-established drawbacks. In particular, consumers usually must accept the software "as is" and trust that the producer has taken the necessary steps to ensure security and reliability. Unfortunately, experience has demonstrated this is not always the case.

How can consumers know if a COTS component has acceptable security and reliability for their needs? A few COTS components provide some information about the development and testing process that was followed. Examples include a few Real-Time Operating Systems (RTOSs) that offer documentation to assist avionics software developers with the DO-178B certification process. But even in these unusual cases, typically only a reduced-functionality version of the RTOS is well documented. For most third-party components, no information about the development and testing process is available.

For organizations developing security or high reliability applications, the inability to assess the quality of third-party components is a significant problem. It is not surprising that one of the earliest proponents of developing better technology for analyzing executables was the National Security Agency, which in 2004 publicly emphasized the importance of tools that analyze binaries[13]. Of particular concern is software used in the nation's critical infrastructure, such as emergency preparedness communications and power plants.

Machine code analysis offers a way to assess third-party code, even when the source is unavailable. The ability to detect defects, vulnerabilities, and intentionally inserted malicious code allows users to regain some control in determining if a piece of software meets their acceptance criteria. Users need not blindly trust the software producer.

Technical advantages of machine code analysis

Source code is not usually provided for COTS software, thus the need for machine code analysis. In fact, even when source code is available, machine code analysis offers many advantages over other analysis techniques. This is because the source code is not executed; rather, it is compiled into a machine code program (the executable). Analyzing programs written in interpreted languages is a different matter, although there, too, the source code is not executed directly on the processor.

Differences may exist between the source code semantics and the compiled executable semantics for several reasons.

This potential mismatch is called the "What You See Is Not What You eXecute" (WYSINWYX) effect[4]. WYSINWYX acknowledges that the semantics in the source code may be incomplete or imprecise in view of what is actually executed in the process.

The WYSINWYX effect can be caused by various factors, including compiler bugs and linking third-party libraries. Figure 1 illustrates how the meaning of the original program can change as modules are added prior to the final executable's creation.

One example of a compiler bug that initiated the WYSINWYX effect was discovered during a 2002 security review at Microsoft[10]. In this case, code like the following appeared in the source for a log-in program:

```
memset(password,'\0', len);

free(password);
```

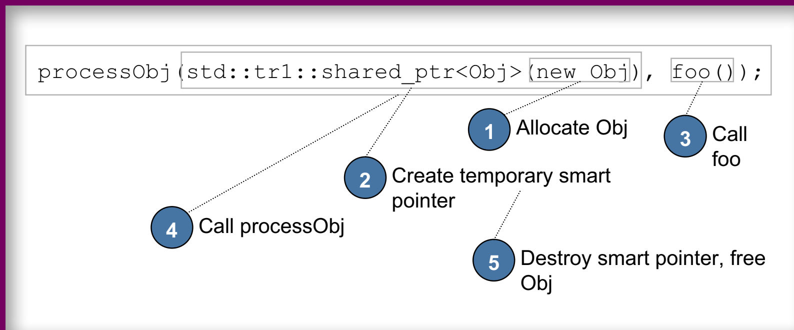
As indicated by its name, the buffer *password* was used to hold a user's password. As a security precaution, the programmer desired to minimize the amount of time this sensitive information was kept in memory. Thus, before deallocating the buffer (line 2), the intent was to overwrite the sensitive password with zeros (line 1).

However, in this case, the Microsoft C++ compiler determined that the password zeroing statement was "useless" and removed it. In a technical sense, the compiler was correct: The zeros written by the *memset* are not supposed to be read by any other statement, and removing the *memset* does not affect the program's results. Nevertheless, the optimization resulted in a security vulnerability that was invisible in the source code.

Every potential WYSINWYX effect underlines machine code analysis tools' advantage over source code analysis tools. The prior section discussed the problem

The importance of step sequence

The ambiguity surrounding function call argument order of evaluation can lead to the WYSINWYX effect. The figure shown here provides an example in C++ from *Effective C++* by Scott Meyers.



The intention of this statement is to:

1. Allocate an Obj on the heap
2. Create a temporary reference-counting pointer (the shared_ptr) to the new Obj
3. Call foo()
4. Pass the shared_ptr and the result of the call to foo() in a call to processObj
5. When processObj returns, the temporary shared_ptr is destroyed, freeing the heap-allocated Obj

The programmer expects these steps to happen in this order. Unfortunately, the compiler is free to reorder the third step before the second (or even the first) step. If the compiler chooses the order 1, 3, 2, a memory leak could occur if the call to foo() throws an exception. Since the smart pointer has not been created at the time of the call to foo(), it will not be destroyed when the exception is thrown, and the Obj will never be deallocated. The compiler's choice – and the potential leak – is evident in the executable, but detecting it in the source code requires considering all potential evaluation orders for the arguments to processObj.

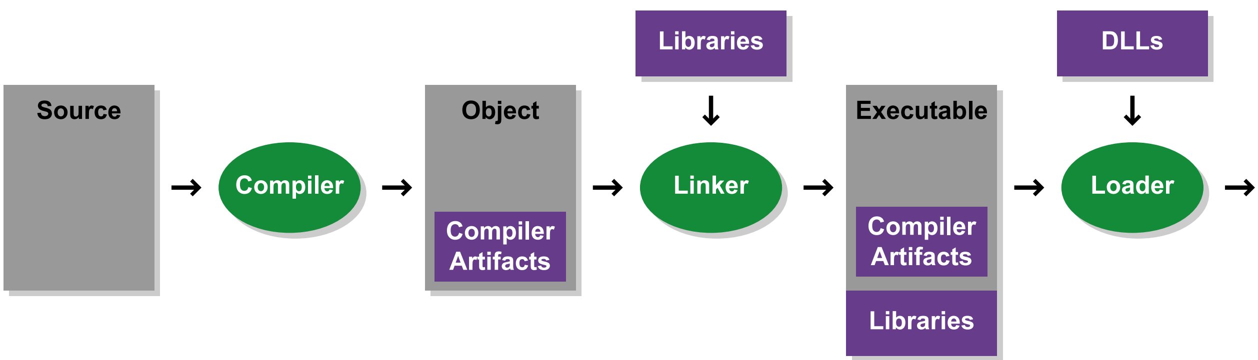


Figure 1

of not having access to a program's source code. However, even developers who have the source rarely have source code for *all* the code eventually included in the executable. Usually, they link their source against third-party libraries that are only in binary form. Especially in the embedded software, source code may include inline assembly. In some cases, modifications are made to the executable after the source is compiled. Source tools usually target programs written in one language, but an executable may be compiled from source in many different languages.

One of the most prominent reasons for the WYSINWYX effect is that source language semantics are usually underspecified. For example, C and C++ do not specify the function call argument order of evaluation. (See the sidebar for an example of this from *Effective C++* by Scott Meyers[14].) Technically, problems due to source language ambiguity are visible in the source code. However, analyzing all the possible behaviors of an ambiguous statement quickly becomes intractable. For this reason, source analysis tools (and often programmers) usually resolve ambiguity by arbitrarily picking one plausible interpretation. Since there is no guarantee that their choice will be the same as the compiler's, language ambiguity is considered to be a major cause of the WYSINWYX effect.

The choices a compiler makes to resolve source language ambiguities can have an important effect on the presence of vulnerabilities. Security exploits frequently rely on details such as data object layout, order of variables on the stack, whether a value is stored in RAM or only in registers, and so on. In a language like C or C++, most of these details are left to the discretion of the compiler.

A source analysis tool cannot consider all the different options a compiler might choose, at least not without making vague approximations. Machine code analysis, however, has the advantage of seeing the exact decisions the compiler made. For this reason, machine code analysis has the potential to be more precise than source code analysis.

Recent advances in machine code analysis

Researchers have made great strides in applying static analysis to machine code. Several groups have demonstrated the utility of machine code analysis for identifying malicious code[6,7,11,12], security vulnerabilities[8], and flaws that affect reliability[3,9].

One use of machine code analysis is to create an *Intermediate Representation* (IR) that captures a program's semantics. Source analysis tools for finding bugs and security vulnerabilities often rely on information (such as types) readily available in source but not machine code. The goal of *IR recovery* is to fill that gap and allow developers to use source analysis techniques on machine code. Compared to developing specialized techniques or adopting source analysis techniques one at a time, IR recovery enables many techniques at once.

One advanced tool for IR recovery from executables is CodeSurfer/x86, which is the result of collaborative research between GrammaTech and the University of Wisconsin. CodeSurfer/x86 is a valuable tool for security analysts who need to understand the potential impact of a piece of malicious code. While the tool currently supports x86 machine code analysis, work on supporting other processor architectures, including PowerPC Architecture and ARM, is underway. Its purpose is to construct an IR similar to those that a compiler or source analysis tool uses. Specifically, the recovered IR represents the following information:

- A disassembly listing
- Control flow graphs, with indirect jumps resolved
- A call graph, with indirect calls resolved
- Information about the program's variables
- Possible values of pointer variables
- Sets of used, killed, and possibly killed variables for each control flow graph node
- Data dependencies, including dependencies between instructions that involve memory access
- Type information (for example, base types, pointer types, and structs)

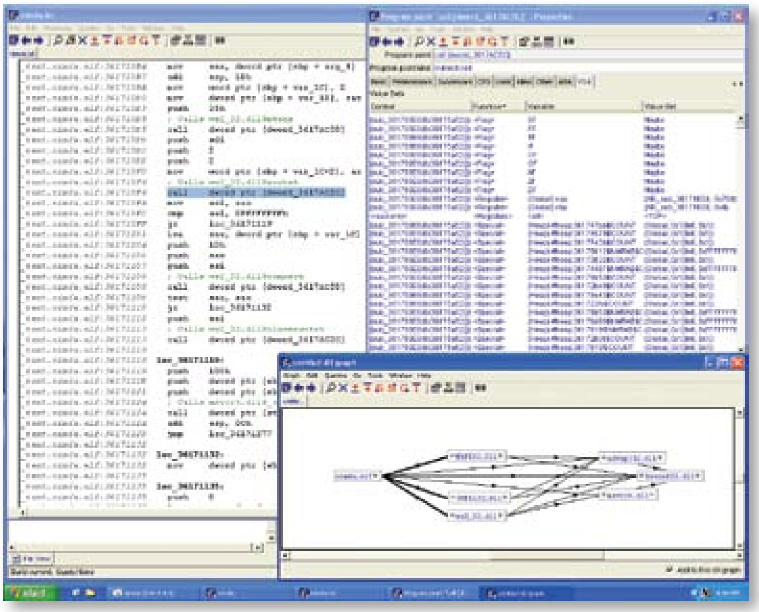


Figure 2

CodeSurfer/x86 performs IR recovery from an executable that runs on an Intel x86 processor. The IR can be used as the basis for building further analyses to find bugs and vulnerabilities or used to browse through a GUI interface. Figure 2 shows the recovered IR for a version of the infamous Nimda virus. The visualized IR components include the disassembly listing, possible data values at chosen program point, and call graph.

Many factors can complicate IR recovery. CodeSurfer/x86 does not rely on symbol table or source code information because such information is often stripped from COTS products. Even if this information was present, it would not be reliable in potentially malicious code. Recovering information about potential pointer values requires analyzing both pointers and numeric values simultaneously because address values and numeric values cannot be easily distinguished[1]. Type information must be inferred based on data access patterns because no structured data types are available[2].

Despite the difficulty in performing IR recovery, the technology has advanced far enough to start producing results. Balakrishnan and Reps recently demonstrated IR recovery use in a Windows device driver analysis[3]. They found that CodeSurfer's IR recovery produces precise results on device drivers and demonstrated that by building on the recovered IR, they could adapt a technique for analyzing device driver source code to analyze machine code and replicate some of the same results[5]. Analyzing the machine code also can help address the WYSINWYX issues discussed earlier.

David Melski is the VP of research at GrammaTech, based in Ithaca, New York. David graduated summa cum laude from the University of Wisconsin in 1994 with a BS in Computer Science and Russian Studies. He received his PhD in Computer Science from the University of Wisconsin in 2002, with research interests in static analysis, profiling, and profile-directed optimization.

GrammaTech
607-273-7340
melski@grammatech.com
www.grammatech.com

Meeting safety-critical needs

Machine code analysis is already playing a valuable role in identifying bugs and security vulnerabilities in software as well as helping users assess third-party code. Safety-critical software producers are expected to start using machine code analysis on their own software to account for the WYSINWYX effect. Both increasing need and increasing tool support and capabilities will continue to drive this growth in machine code analysis. ECD

References

[1] Balakrishnan, G. and T. Reps. Analyzing Memory Accesses in x86 Executables. International Conference on Compiler Construction. 2004. Barcelona, Spain: Springer Verlag. pp. 5-23.

[2] Balakrishnan, G. and T. Reps, DIVINE: DIscovering Variables IN Executables. VMCAI. 2007. Nice, France.

[3] Balakrishnan, G. and Reps, T., Analyzing stripped device-driver executables. TACAS, 2008.

[4] Balakrishnan, G., T. Reps, D. Melski, and T. Teitelbaum. WYSINWYX: What You See Is Not What You eXecute. Proc. IFIP Working Conference on Verified Software: Theories, Tools, Experiments (to appear). 2005. Zurich, Switzerland.

[5] Ball, T. and S.K. Rajamani. The Slam Project: Debugging System Software via Static Analysis, http://research.microsoft.com/slam/papers/popl02.pdf.

[6] Christodorescu, M., S. Jha, D. Maughan, D. Song, and C. Wang, eds. Malware Detection. *Advances in Information Security*, ed. S. Jajodia. Vol. 27. 2007, Springer.

[7] Christodorescu, M., S. Jha, S.A. Seshia, D. Song, and R.E. Bryant. Semantics-Aware Malware Detection. IEEE Symp. on Security and Privacy. 2005. Oakland, CA.

[8] Ganapathy, V., S.A. Seshia, S. Jha, T. Reps, and R.E. Bryant. Automatic discovery of API-level exploits. International Conference on Software Engineering. 2005.

[9] Godefroid, P., M.Y. Levin, and D. Molnar. Automated Whitebox Fuzz Testing. 2007, Microsoft Research MSR-TR-2007-58.

[10] Howard, M. Some Bad News and Some Good News, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dncode/html/secure10102002.asp.

[11] Kinder, J., S. Ktzenbeisser, C. Schallhart, and H. Veith. Detecting Malicious Code by Model Checking. Conference on Detection of Intrusions and Malware and Vulnerability Assessment. 2005.

[12] Kruegel, C., W. Robertson, and G. Vigna. Detecting Kernel-Level Rootkits Through Binary Analysis. Annual Computer Security Applications Conference. 2004.

[13] Leveraging Cybersecurity – Interview with Daniel G. Wolf, Information Assurance Director, National Security Agency. *Military Information Technology*, Online Edition. February 9, 2004.

[14] Meyers, S. *Effective C++*, Third Edition. 2005: Addison-Wesley.

Embedded Linux file system management: Tooling for success

By Troy Kitch and Joe Green

As budgets decrease, product life cycles shorten, and more features are packed into electronic devices, developers are feeling the pressure to deliver advanced software. While Linux and open source are often the right choices for embedded development, they cannot work magic. Some costs are still associated with selecting open source development tools. In deciding which tools to use, developers must consider if they are spending their time on that which is truly innovative and differentiating or on busywork such as integration and support.

Tool selection can help or hinder developers' efforts to gain control of a project. The litmus test for truly valuable tools is whether or not they assist developers to enable the right features within a product's short life cycle.

One area of embedded Linux development subjected to this loss of control is the process of building the often complex file systems Linux depends upon. Platform developers need to integrate and install dozens and sometimes hundreds of separate software components, but creating a target file system by hand is time-consuming, difficult, and complex. Without the right tools, this process can be prone to error. That time and effort can be better spent on developing features that differentiate products from the competition. Among the most difficult and, in a sense, least rewarding activities is Linux file system configuration.

Traditional Linux development

The first incarnations of Linux were developed in an environment where the target was either on the host itself or a comparable machine. Because of this, accompanying tools traditionally have been tailored to a host development environment. Fundamentally, it is easier for developers to develop for a machine similar to the machine they are developing on. As the target environment draws farther away from the host environment, developers face greater challenges.

Developing for the host or similar environment does not require much extra work. In fact, some embedded systems are so similar to a PC that developers can run the actual Linux distribution of their choice, such as SuSe or Fedora. But developing for one of the different processor architectures in the market today can be a challenge.

Embedded Linux development and cross-development

Environments become more complex when resources are constrained and the target processor is running one of the many non-x86 processors available. In this environment, it is more productive to develop on a host machine with many resources and then cross-compile to develop a binary for the particular processor architecture.

Because of its modular design, Linux runs efficiently on small devices. Developers can select the features they want to implement and prune the ones they don't need. True finesse is required to accomplish this task and fine-tune the features and user interface for a particular device.

But just because it is modular doesn't mean there is a clear path to success. Developers must consider two approaches: follow a package-based installation, which leverages the majority of Linux distributions available, or integrate packages into a build system environment. One issue with a build system is that it is incongruent to the way Linux is distributed – generally, as packages like .deb or .rpm. Another challenge is the learning curve necessitated by any particular build system. The following discussion will present package managers and explain how they can help or hurt developers' ability to build embedded Linux file systems.

Choosing an Embedded Linux file system manager

Ian Murdock, founder of the Debian GNU/Linux distribution, describes package management as "the single biggest advancement Linux has brought to the industry." He thinks package management blurs the boundaries between Operating Systems (OSs) and applications, making it "easier to push new innovations … into the marketplace … and evolve the OS."

The tools available for building embedded Linux file systems become more useful in cross-platform and resource-constrained environments. It is important for developers to ask a few questions before selecting a particular tool or Integrated Development Environment (IDE). Consider: Where are you getting your Linux source, and how is it wrapped up for delivery? What are you using the file system for, and how can it be optimized for a particular target? The package installer that developers choose can make the difference between an optimized file system and a poor excuse for an embedded device.

A common approach to managing an embedded Linux file system is using a package manager like RPM or dpkg for installing and removing a nonroot directory, fakeroot for a sneaky way of doing *chroot* that doesn't require root permission, or development with a virtual target like the Linux QEMU processor emulator or Virtutech's Simics Virtual Platform. Each of these different options has pros and cons that may be remedied by an alternative solution: a platform image builder.

Package manager

The reigning champion of file system management is the RPM package manager. RPM is useful because it is standard and available on most Linux systems. Like others of its ilk, RPM has built-in capability to install to a directory that is not the system root, which is advantageous because smart developers don't want the ability to accidentally destroy their host file system. But RPM doesn't solve package dependencies; it merely determines whether or not dependencies are met. It requires that developers already know the dependencies and therefore won't go checking the various packages to determine which ones need to be resolved to build the file system.

To resolve dependencies, developers must add another tool such as Yellowdog Updater (*yum*) or Advanced Packaging Tool (*apt*). As a library of C++ functions, apt was created to efficiently install packages while handling dependencies and taking care of configuration files during upgrades. However, apt and yum are limited to the command line. If developers are new to Linux or these tools in particular, they may face a relatively steep learning curve. Additionally, all information is textual and often difficult to navigate relative to graphical file system managers.

Faking out root

Linux can set permissions appropriately to protect the system from malicious intent, ignorance, or forgetfulness associated with modifying or deleting files. Linux has a base root directory that houses all subdirectories. This base root, often called "/" (pronounced slash), is protected from Linux system users. Administrators limit access to "/" and provide each user with a modifiable home subdirectory underneath it. This enables users to access the system and accomplish their tasks with the appropriate rights and abilities.

However, some file system creation operations require root privilege. Without root privilege, developers can't create a file owned by another user, create a device node, or commit a change root operation. These restrictions limit the ability to configure a file system.

One option is to use a tool like *fakeroot*, which can create a virtual "/" file system tree within a directory on a user's host system. It provides a fake "/" environment by redefining standard functions within the host system libraries. In this way it changes the utilities that reference files and captures privileged information about files without requiring root privilege to create them. Additionally, it can be used with standard utilities without requiring special tooling. As with RPM, fakeroot does not solve dependencies, leaving this arduous task up to the developer.

And here's an additional challenge involved with both RPM and fakeroot: Developers may want to execute code within the target directory, and if that code is for a different architecture than the host, it simply won't run.

Virtual development

A third alternative to building a file system is using a virtual environment like QEMU, Simics, or VMware Workstation. This solves the challenge that both package managers and fakeroot tools face – not being able to run code on a host with a different architecture than the target. A virtual environment also provides the ability to develop for a virtual target using all the resources available on the host, such as memory, storage, and a fast processor, enjoying many of the advantages of self-hosted development on a PC.

Using a virtual environment is often faster than developing on the target itself, but it can add complexity, slowing processor power relative to cross-development.

Platform image builder

As outlined in the beginning of this article, platform developers need to integrate and install dozens if not hundreds of separate software components, but creating a target file system by hand is time-consuming, difficult, and complex. Once the file system has been created, it must be converted into a target image (see sidebar). More advanced tools can simplify the task of assembling, tuning, and creating an image of the file system.

Platform image builder accomplishes this by providing a visual map of the system for selecting Linux target packages, integrating custom packages and kernels, dynamically determining file system size, automatically resolving dependencies and conflicts, and generating file systems in several standard formats.

Platform image builder is useful for trimming components from a final image, whether they are individual files or whole hierarchies (for example, documentation and sample configuration files). Having the ability to visibly sort packages into groups like documentation, fonts, graphics, or interpreters can provide developers

Converting a file system into a target image

Developers have three options available to create a target image once a file system tree is laid out:

- Use *tar* or *cpio* to package the files into an archive and then transfer to the target and unpack
- Use a utility program provided by some file system types (JFFS2, CRAMFS) to create a file system image from a directory tree (QEMU creates a disk image from a tree)
- Use *loopback* to mount a file directly and initialize and copy files into the image using standard utilities (requires root privilege)

with quick access to and faster elimination of unnecessary packages (see Figure 1).

This ability also enables users to drill down and sort through files and directories via a tree hierarchy. To remove a file, simply uncheck the adjacent box (Figure 2). In addition to removing individual files, developers may choose to mark entire required supporting packages as “phantom.” Such packages are necessary for building other packages but not for the final build, and are therefore not included at runtime.

While setting up a platform image project (.pib), developers can include and integrate custom packages and kernels. This flexibility provides the competitive differentiation and control embedded developers demand in such a fast-paced market.

Platform image builder gives a number of options affecting the size of the image file. It does this by:

1. Optimizing library footprints and reducing the size of some shared libraries
2. Pre-linking executable files, making start-up times faster
3. Removing debugging symbols from final binaries, making them both smaller and faster

These measures are designed not only to reduce the size of the image, but also to improve performance.

Platform image builder uses the dependency information included in RPM packages to automatically include supporting packages as necessary (Figure 3). This is consistent with the way Linux is generally distributed and takes away the mundane tasks of manually working out dependencies. However, when making package selections, it is possible to select packages that conflict with each other. Since this results in errors and prevents projects from building successfully, developers must resolve conflicts before creating images. Platform image builder helps by flagging and listing all conflicts as errors, making it easy to review and change them until all conflicts are removed.

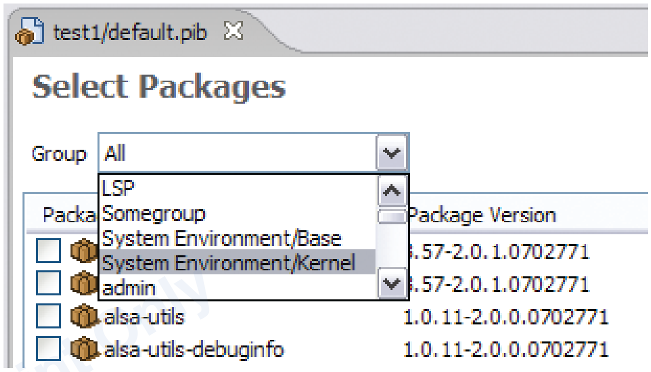


Figure 1

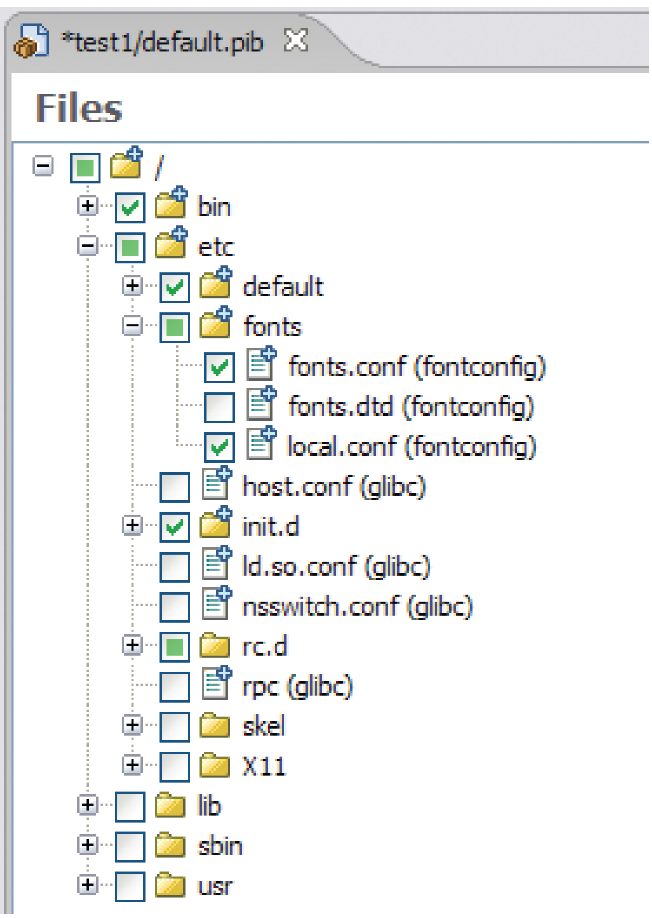


Figure 2

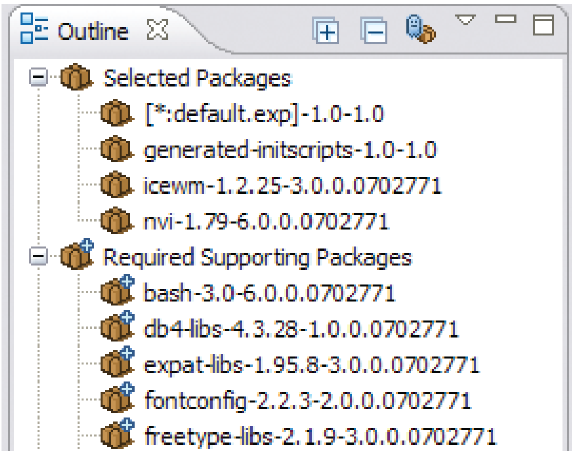


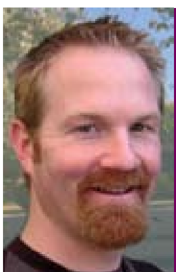
Figure 3

Finally, with a file system management tool like platform image builder, platform developers can produce common file formats, including ext2, JFFS2, cpio, CRAMFS, and ext3. Depending on the format, different options are available to configure the image and set mount points.

In contrast to the previously mentioned methods of creating and managing file systems for embedded Linux development, platform image builder has a graphical user interface that does not require an emulated environment and therefore doesn't use as many resources relative to a virtual machine. Platform image builder turns file systems into an image without the complexity and slower processing power, giving developers the productivity required for embedded Linux cross-development projects.

The tools litmus test

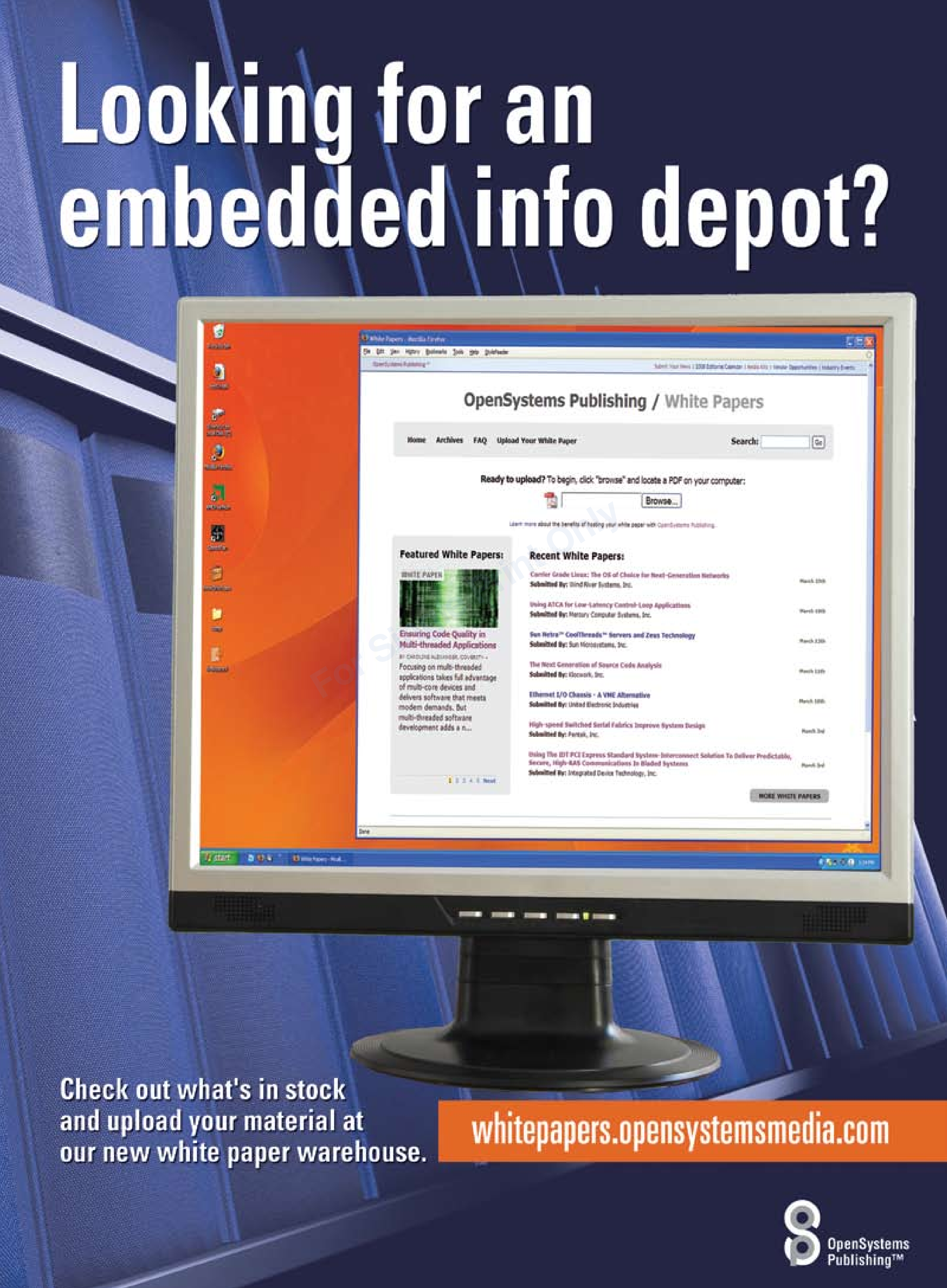
Embedded Linux developers can obtain more control over building complex file systems with a tool like platform image builder, which makes assembling, tuning, and creating a file system image easier to accomplish. This enables developers to spend time developing features that differentiate products from the competition. Remember, the litmus test for truly valuable tools is whether or not they assist developers to enable more of the right features within a product's short life cycle. **ECD**



Troy Kitch *is senior product manager of the developer tools team at MontaVista Software, based in Santa Clara, California. Troy has spent more than a decade in the development and security software industries, focused on developer productivity, data storage, and disaster recovery. At MontaVista, Troy is responsible for helping organizations get the most out of open source by managing the MontaVista DevRocket integrated development environment. Troy has a BS in Agribusiness from Cal Poly, San Luis Obispo.*

Joe Green *is manager of the developer tools team at MontaVista Software. Joe has been working with Linux and UNIX systems at companies such as MontaVista Software, IBM, and SGI for 20 years and has been happily using Linux since version 0.12. He is particularly fond of kernel and graphics code as well as real-time and embedded systems. He has a BSEE from the University of Miami.*

MontaVista Software
408-572-8000
info@mvista.com
www.mvista.com




Storage Leader !
.Integration
.Reliability
.Performance
New!
For Single Print Only

UNIT UNDER TEST

Consumer electronics testing brings standards to life

By Don Dingee

Testing today's consumer electronics means spending a lot of time up front validating the design. The proliferation of standards in consumer electronics has made interoperability better but testing more challenging. Test and instrumentation vendors are creating tools to test compliance and interoperability against specifications, helping designers cope with complex interfaces such as high-speed serial interconnects and higher-level protocols.

Some specifications are tailored expressly to consumer electronics devices. The Mobile Industry Processor Interface (MIPI) Alliance (www.mipi.org) has created several specifications focused on the interconnect between functional blocks in a mobile device. The Camera Serial Interface (CSI-2) is a low pin count, high-speed serial interface between the camera silicon and the device processor. Similarly, the Display Serial Interface (DSI) connects the LCD display to the processor. Each specification defines a D-PHY with signaling characteristics and protocols supporting data rates as high as 1 Gbps.

Jean Manuel Dassonville, product manager for digital wireless test products at Agilent Technologies, states that, "Interfaces like the MIPI D-PHY variants are difficult to analyze with a general-purpose signal analyzer." Instead of just looking at signals, packet-based analysis is needed. Additionally, both stimulus (pattern generation) and analysis (decoding traffic) are helpful to designers.

Agilent has taken the approach of putting the specifics of stimulus and analysis into probe units, allowing a logic analyzer mainframe to support various functions. For MIPI D-PHY testing, Agilent offers the N4851A Digital Acquisition Probe (Figure 1) and the N4861A 800 Mbps Stimulus Probe, both of which connect to a 16800 or 16900 logic analyzer. Other probe solutions include the N4850 and N4860 for DigRF v3, an interface between the RF and baseband blocks. Designers can mix and match probes to suit their testing needs.


Figure 1

LeCroy has gone after a similar problem for more familiar standards: Ethernet, USB, PCI Express, Serial ATA (SATA), and Ultra-Wideband (UWB). The company's QualiPHY automated serial data compliance test framework offers guided test setup and automated measurement capability for high-speed serial interface compliance testing using an oscilloscope outfitted with an acquisition platform and software.


Figure 2

For example, the LeCroy SDA 11000 Serial Data Analyzer with the QualiPHY UWB package (Figure 2) can measure UWB signals in all six band groups, with software performing modulation analysis, such as quadrature phase shift keying and dual carrier modulation, constellation display, in-phase and quadrature versus time display, and magnitude phase versus tone display. Similarly, the SDA 11000 with the QualiPHY SATA package can handle Gen 2 SATA testing.

Protocol testing is becoming much more sophisticated as well. Ixia has a suite of offerings for IP test automation. From the basics of IPv6 through higher levels of triple play infrastructure, Ixia's tools use deep packet inspection and real-world traffic emulation to provide the robust testing needed (see Figure 3). The IxLoad platform handles a variety of protocols over thousands of endpoints combining voice, video, and data traffic.

As interfaces in consumer electronics devices and networks have become more sophisticated, testing platforms have responded with better solutions. Trying to home brew testing products for these complex interfaces simply isn't necessary with the variety of solutions available today.

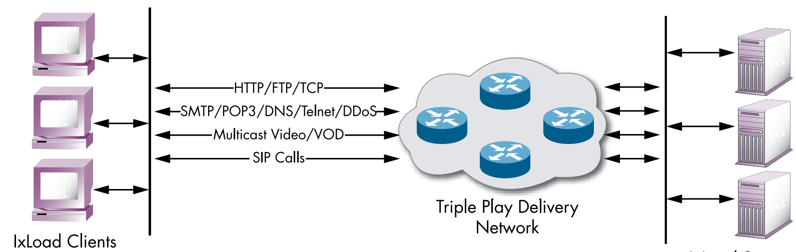
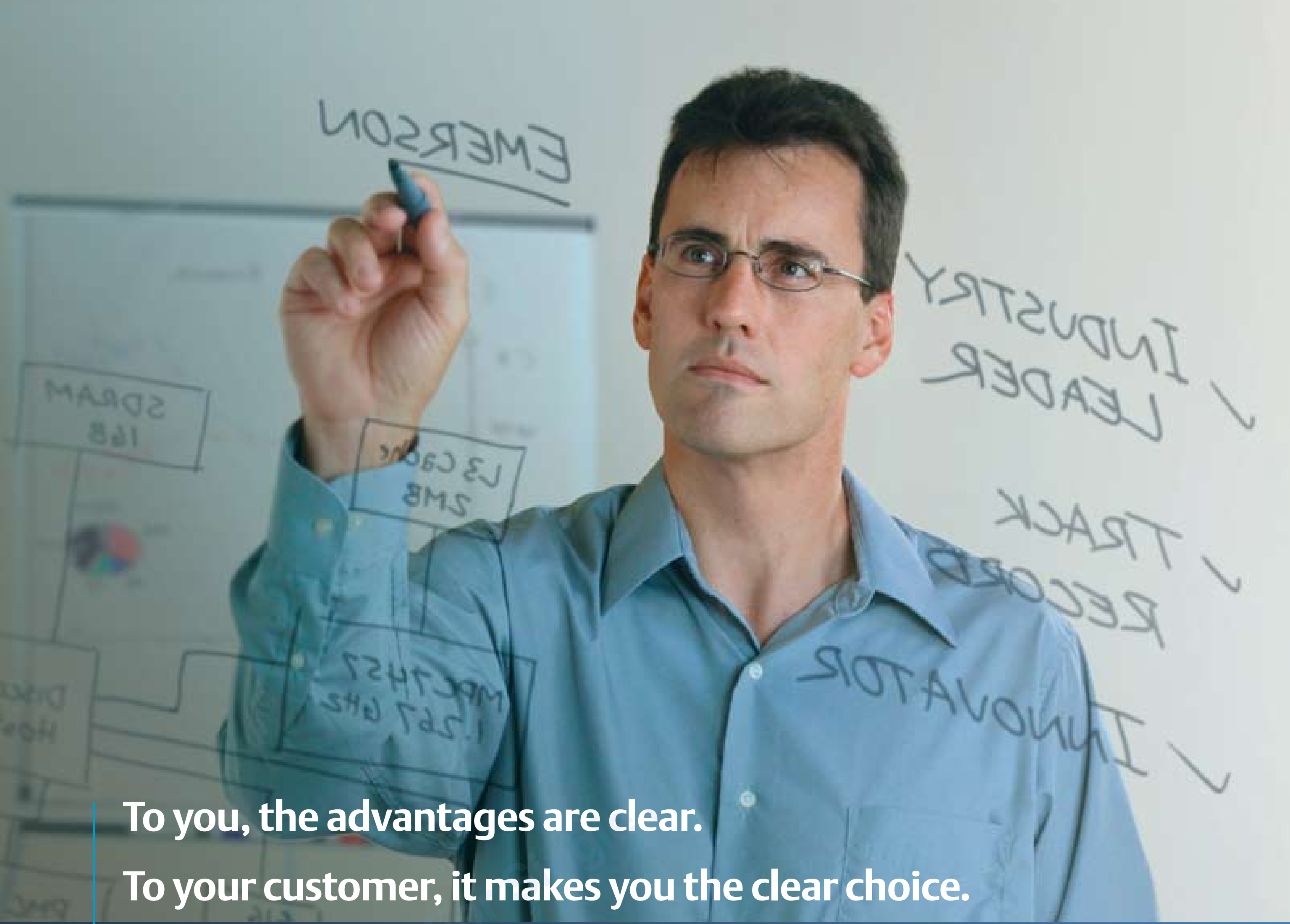


Figure 3



To you, the advantages are clear.

To your customer, it makes you the clear choice.

Emerson Network Power is now clearly the leading provider of embedded computing solutions.

From platforms, blades and modules, to software and services, Emerson's industry-leading technology portfolio is ready to help solve your customers' most demanding applications. Make our AdvancedTCA®, MicroTCA™, AdvancedMC™, CompactPCI®, Processor PMC, VMEbus and OpenSAF™ standards-based products your first choice.

See how Emerson Network Power can help you build a clear advantage.

Go to www.EmersonNetworkPower.com/EmbeddedComputing

Standards-based Embedded Computing

Just another reason why Emerson Network Power is the global leader in enabling Business-Critical Continuity™

The Embedded Communications Computing business of Motorola is now a business of Emerson Network Power.

EMERSON™
Network Power

EMERSON. CONSIDER IT SOLVED.™

Advertiser Information

OpenSystems Publishing

Advertising/Business Office

30233 Jefferson Avenue
St. Clair Shores, MI 48082
Tel: 586-415-6500 ■ Fax: 586-415-4882

Vice President Marketing & Sales
Patrick Hopper
phopper@opensystems-publishing.com

Business Manager
Karen Layman

Sales Group

Dennis Doyle
Senior Account Manager
ddoyle@opensystems-publishing.com

Tom Varcie
Senior Account Manager
tvarcie@opensystems-publishing.com

Doug Cordier
Account Manager
dcordier@opensystems-publishing.com

Andrea Stabile
Advertising/Marketing Coordinator
astabile@opensystems-publishing.com

Christine Long
E-marketing Manager
clong@opensystems-publishing.com

Regional Sales Managers

Jerry Bleich
New England
jbleich@opensystems-publishing.com

Barbara Quinlan
Midwest/Southwest
bquinlan@opensystems-publishing.com

Ron Taylor
East Coast/Mid Atlantic
rtaylor@opensystems-publishing.com

Ernest Godsey
Central and Mountain States
egodsey@opensystems-publishing.com

Denis Seger
Southern California
dseger@opensystems-publishing.com

Sydele Starr
Nothern California
sstarr@opensystems-publishing.com

International Sales

Dan Aronovic
Account Manager – Israel
daronovic@opensystems-publishing.com

Sam Fan
Account Manager – Asia
sfan@opensystems-publishing.com

Reprints and PDFs

Nan Lamade: 800-259-0470
ecdreprints@opensystems-publishing.com


Your potential. Our passion.®
Microsoft®