UBM Electronics

Learn today. Design tomorrow.

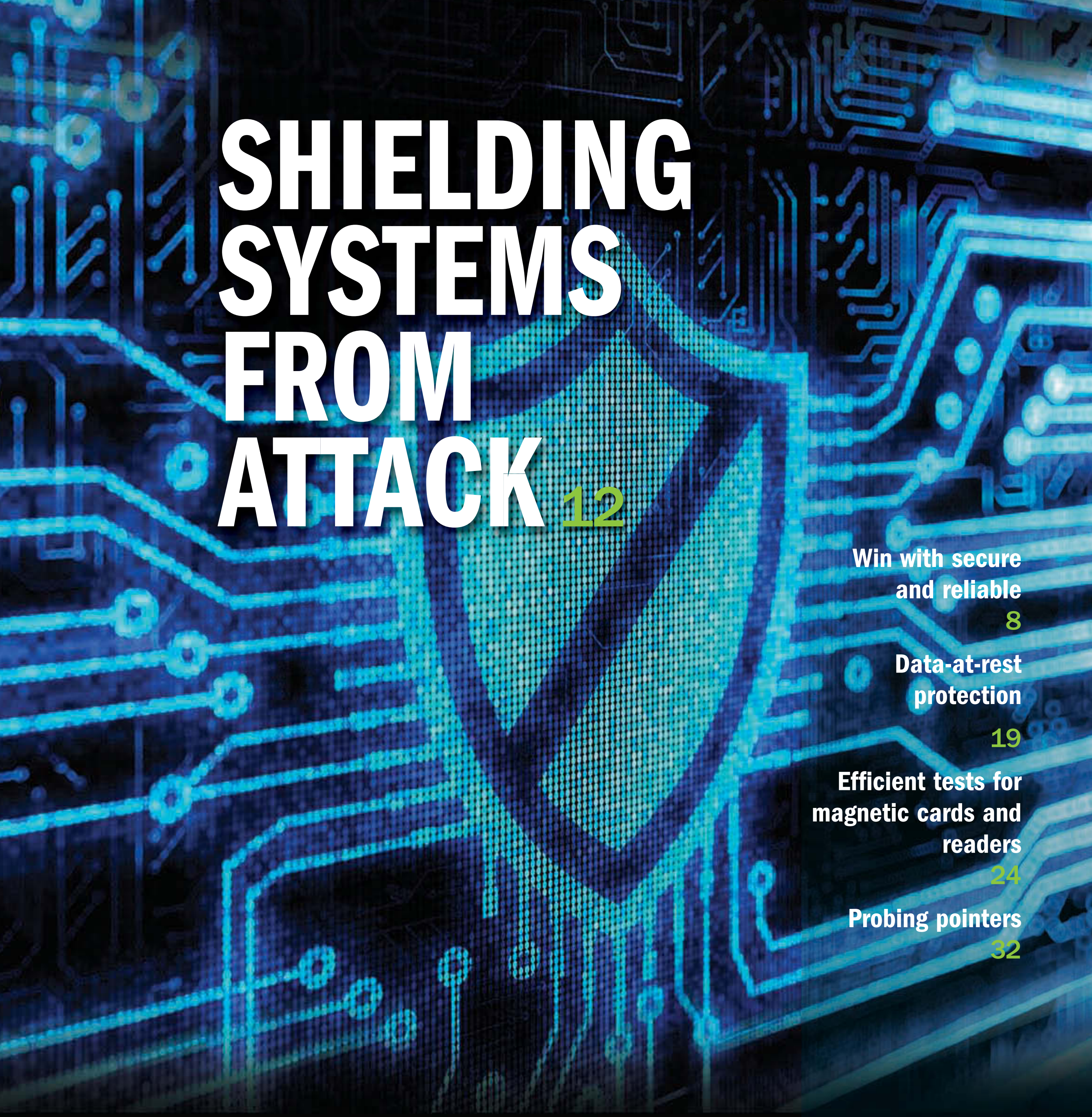
ESD

EMBEDDED SYSTEMS DESIGN

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

VOLUME 25, NUMBER 3

APRIL 2012



DesignNews

Digi-Key CORPORATION CONTINUING EDUCATION CENTER

Get current, without the hassle.

With Digi-Key's Continuing Education Center, catch up to 180 days of free, interactive courses engineered by Design News, starting January 2012. Build your engineering know how. Take one or all. It's your choice.

Tracks include: Microcontrollers Basic, Microcontrollers Advanced, MEMS Sensor Technology, Lighting, Embedded Internet and Cloud, Wirelesss, and Power.

REGISTER AT
DesignNews.com/dkcec

FOLLOW US

@DigiKeyCEC

Digi-Key Continuing Education Center

UBM Electronics

TRUSTED SOFTWARE FOR DEFENSE

For nearly 30 years the world's leading defense companies have trusted **Green Hills Software's** secure and reliable high performance software for mission-critical and safety-critical applications.

From the Joint Tactical Radio System, through naval anti-surface missiles, to the F-35 Lightning II and Eurofighter Typhoon, Green Hills Software has been delivering proven and secure underpinning technology.

To find out how the world's most secure and reliable operating system and software can take the risk out of your defense project, visit **www.ghs.com/s4d**

Green Hills® SOFTWARE

Scan here to Find TI.

mouser.com
Semiconductors and electronic components for design engineers.

Authorized Distributor

Your Complete Source for Everything TI.

Over **80,000** Searchable Part Numbers. More than **100** TI Product Knowledge Centers.
Find TI here. Faster at mouser.com.

mouser.com/ti
The Newest Products for Your Newest Designs®

TEXAS INSTRUMENTS

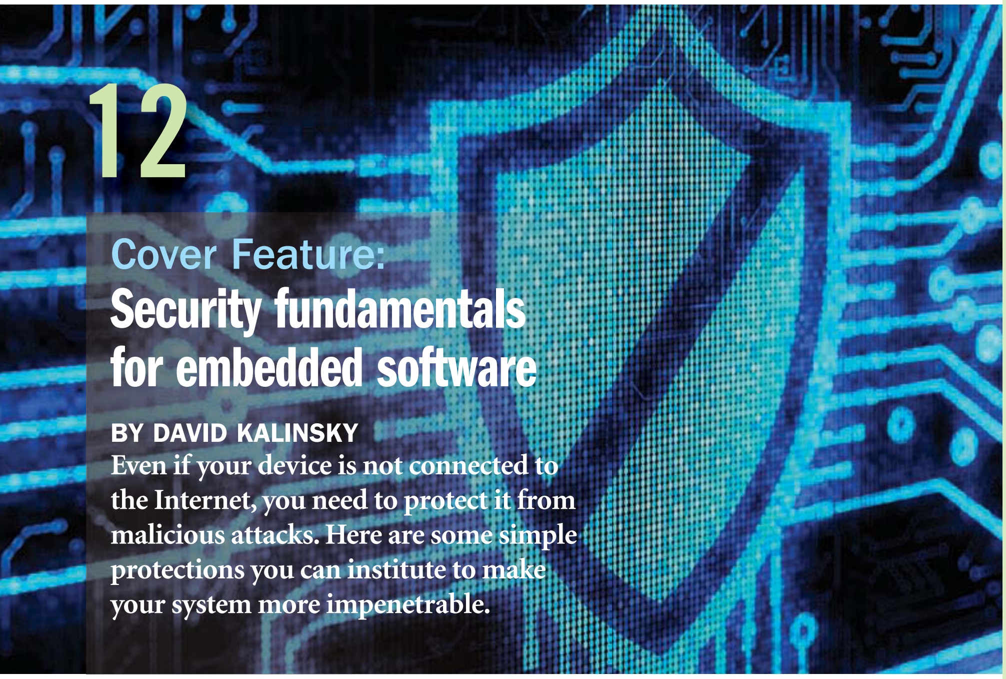
MOUSER ELECTRONICS®
a tti company

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

Learn today. Design tomorrow.

ESD EMBEDDED SYSTEMS DESIGN

VOLUME 25, NUMBER 3
APRIL 2012



EMBEDDED SYSTEMS DESIGN (ISSN 1558-2493) print; (ISSN 1558-2507 PDF-electronic) is published 10 times a year as follows: Jan/Feb, March, April, May, June, July/August, Sept., Oct., Nov., Dec. by the EE Times Group, 303 Second Street, Ste 900, San Francisco, CA 94107, (415) 947-6000. Please direct advertising and editorial inquiries to this address. SUBSCRIPTION RATE for the United States is $55 for 10 issues. Canadian/Mexican orders must be accompanied by payment in U.S. funds with additional postage of $6 per year. All other foreign subscriptions must be prepaid in U.S. funds with additional postage of $15 per year for surface mail and $40 per year for airmail. POSTMASTER: Send all changes to EMBEDDED SYSTEMS DESIGN, EE Times/ESD, PO Box #3609, Northbrook, IL 60065-3257, embedsys@omeda.com. For customer service, telephone toll-free (847) 559-7597. Please allow four to six weeks for change of address to take effect. Periodicals postage paid at San Francisco, CA and additional mailing offices. EMBEDDED SYSTEMS DESIGN is a registered trademark owned by the parent company, UBM Electronics. All material published in EMBEDDED SYSTEMS DESIGN is copyright © 2012 by UBM Electronics. All rights reserved. Reproduction of material appearing in EMBEDDED SYSTEMS DESIGN is forbidden without permission.

AVNET® Embedded
Accelerating Your Success™

THREADX: WHEN IT REALLY COUNTS

When Your Company's Success, And Your Job, Are On The Line - You Can Count On Express Logic's ThreadX® RTOS

THREADX

Express Logic has completed 14 years of successful business operation, and our flagship product, ThreadX, has been used in over 1 billion electronic devices and systems, ranging from printers to smartphones, from single-chip SoCs to multiprocessors. Time and time again, when leading manufacturers put their company on the line, when their engineering team chooses an RTOS for their next critical product, they choose ThreadX.

Our ThreadX RTOS is rock-solid, thoroughly field-proven, and represents not only the safe choice, but the most cost-effective choice when your company's product simply must succeed. Its royalty-free licensing model helps keep your BOM low, and its proven dependability helps keep your support costs down as well. ThreadX repeatedly tops the time-to-market results reported by embedded developers like you. All the while, Express Logic is there to assist you with enhancements, training, and responsive telephone support.

Join leading organizations like HP, Apple, Marvell, Philips, NASA, and many more who have chosen ThreadX for use in over 800 million of their products – because their products are too important to rely on anything but the best. Rely on ThreadX, when it really counts!

Contact Express Logic to find out more about our ThreadX RTOS, FileX® file system, NetX™ Dual IPv4/IPv6 TCP/IP stack, USBX™ USB Host/Device/OTG stack, and our PEGX™ graphics toolkit for embedded GUI development. Also ask about our TraceX® real-time event trace and analysis tool, and StackX™, our patent-pending stack size analysis tool that makes stack overflows a thing of the past. And if you're developing safety-critical products for aviation, industrial or medical applications, ask about our new Certification Pack™ for ThreadX.

expresslogic

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

EMBEDDED SYSTEMS DESIGN

Director of Content
Colin Holland
colin.holland@ubm.com

Managing Editor
Susan Rambo
(415) 947-6675
susan.rambo@ubm.com

Acquisitions/Newsletter Editor, ESD and Embedded.com
Bernard Cole
(928) 525-9087
bccole@acm.org

Contributing Editors
Michael Barr
Jack W. Crenshaw
Jack G. Ganssle
Dan Saks

Art Director
Debee Rommel
debee.rommel@ubm.com

Production Director
Donna Ambrosino
dambrosino@ubm-us.com

Article submissions
After reading our writer's guidelines, send article submissions to Bernard Cole at
bccole@acm.org

Subscriptions/RSS Feeds/Newsletters
www.eetimes.com/electronics-subscriptions

Subscriptions Customer Service (Print)
Embedded Systems Design
PO Box # 3609
Northbrook, IL 60065- 3257
embedsys@omeda.com
(847) 559-7597

Article Reprints, E-prints, and Permissions
Mike Lander
Wright's Reprints
(877) 652-5295 (toll free)
(281) 419-5725 ext.105
Fax: (281) 419-5712
www.wrightsreprints.com/reprints/index.cfm ?magid=2210

Publisher
David Blaza
(415) 947-6929
david.blaza@ubm.com

Associate Publisher, *ESD* and *EE Times*
Bob Dumas
(516) 562-5742
bob.dumas@ubm.com

UBM Electronics

Corporate—UBM Electronics

Paul Miller	*Chief Executive Officer*
David Blaza	*Vice President*
Karen Field	*Senior Vice President, Content*
Felicia Hamerman	*Vice President, Marketing*
Brent Pearson	*Chief Information Officer*
Jean-Marie Enjuto	*Vice President, Finance*
Amandeep Sandhu	*Director of Audience Engagement & Analytics*
Barbara Couchois	*Vice President, Partner Services & Operations*

Corporate—UBM LLC

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

A tale of two design sites

Some of you will undoubtedly be reading this while attending the ESC/DESIGN West event or in the lead up to it. For those not fortunate to be joining us in San Jose in the last week of March, I would like to highlight what we'll be providing through the event website *www.ubmdesign.com*.

As well as all the information required for people to plan a conference visit, we'll be updating news from the show via the Breaking News tab on *www.ubmdesign.com* (or directly here: *www.ubmdesign.com/breaking-news*). What's more exciting is that you'll be able to access several areas on the site aimed at extending the life of content from the shows and well as providing two-way interaction. One such section is the forum-like Answerstream (*www.ubmdesign.com/answerstream*), which will provide discussion areas on:

Analog/mixed signal
Android
Development tools
FPGA
LEDs
Low power
Memory
Medical
Microprocessors/microcontrollers
RTOS
Security
Sensors
Test and measurement

This is part of our goal to provide a continuing educational experience and community resource year-round. The web site will provide intelligent, personalized content in real-time including white papers, webinars, videos, special offers, and collateral. The intelligent content functionality not only delivers targeted subject matter content but also dynamically monitors your reactions and engagement patterns in able to recommend additional, meaningful content. The application is platform agnostic and available to work on tablet and desktop devices.

In parallel over the coming months, we'll be looking to upgrade and enhance our sister website, Embedded.com. This site has long been regarded as the must-visit location for many people involved in the embedded sector worldwide. I'm extremely keen to receive feedback from existing and potential readers on what you would like to see from Embedded.com—let me know what we do well, the content you like and would like to see expanded, but more importantly where we could improve our service to you.

The design of embedded systems is continually evolving and we want Embedded.com to reflect that while making sure we don't "throw the baby out with the bathwater" and discard anything you find useful and rely on to help you to implement your designs. What else would be useful to you—more code for download, detailed descriptions of development kits? What else should we add that could make you more efficient?

I look forward to receiving comments from as many of you as possible.

In the meantime, live at ESC and later in April we'll be doing a webinar on the results of our annual Embedded Market Survey—keep an eye on Embedded.com and its newsletter for an update on the date.

Colin Holland
Colin.holland@ubm.com

Colin Holland is the director of content for Embedded Systems Design magazine, *Embedded.com, and the DesignWest and East (which includes the Embedded Systems Conferences). You may reach him at colin.holland@ubm.com.*

By Michael Barr

Building reliable and secure embedded systems

In this era of 140 characters or less, it has been well and concisely stated that, "*reliability* concerns *accidental* errors causing failures, whereas *security* concerns *intentional* errors causing failures." In this column, I expand on this statement, especially as regards the design of embedded systems and their place in our network-connected and safety-concious modern world.

As the designers of embedded systems, the first thing we must accomplish on any project is to make the hardware and software work. That is to say, we need to make the system behave as it was designed to. The first iteration of this is often flaky; certain uses or perturbations of the system by testers can easily dislodge the system into a non-working state. In common parlance, "expect bugs."

Given time, tightening cycles of debug and test can get us past the bugs and through to a shippable product. But is a debugged system good enough? Neither reliability nor security can be tested into a product. Each must be designed in from the start. So let's take a closer look at these two important design aspects for modern embedded systems and then I'll bring them back together at the end.

Neither reliability nor security can be tested, debugged, or patched into a product. They must be designed into embedded systems from day one.

RELIABLE EMBEDDED SYSTEMS

A product can be stable yet lack reliability. Consider, for example, an anti-lock braking computer installed in a car. The software in the anti-lock brakes may be bug-free, but how does it function if a critical input sensor fails?

Reliable systems are robust in the face of adverse run-time environments. Reliable systems are able to work around errors encountered as they occur to the system in the field—so that the number and impact of failures are minimized. One key strategy for building reliable systems is to eliminate single points of failure. For example, redundancy could be added around that critical input sensor—perhaps by adding a second sensor in parallel with the first.

Another aspect of reliability that is under the complete control of designers (at least when they consider it from the start) are the "fail-safe" mechanisms. Perhaps a suitable but lower-cost alternative to a redundant sensor is detection of the failed sensor with a fall back to mechanical braking.

Failure Mode and Effect Analysis (FMEA) is one of the most effective and important design processes used by engineers serious about designing reliability into their systems. Following this process, each possible failure point is traced from the root failure outward to its effects. In an FMEA, numerical weights can be applied to the likelihoods of each failure as well as the seriousness of consequences. An FMEA can thus help guide you to a cost effective but higher reliability design by highlighting the most valuable places to insert the redundancy, fail-safes, or other elements that reinforce the system's overall reliability.

In certain industries, reliability is a key driver of product safety. And that is why you see these techniques,

Michael Barr is CTO of Barr Group and a leading expert in the architecture of embedded software for secure and reliable real-time computing. Barr is also a former lecturer at the University of Maryland and Johns Hopkins University and author of three books and more than sixty five articles and papers on embedded systems design. Contact him at mbarr@barrgroup.com.

FMEA, and other design-for-reliability processes being applied by the designers of safety-critical automotive, medical, avionics, nuclear, and industrial systems. The same techniques can, of course, be used to make any type of embedded system more reliable.

Regardless of your industry, it is typically difficult or impossible to make your product as reliable via patches. There's no way to add hardware like that redundant sensor, so your options may reduce to a fail-safe that is helpful but less reliable overall. Reliability cannot be patched or tested or debugged into your system. Rather, reliability must be designed in from the start.

SECURE EMBEDDED SYSTEMS

A product can also be stable yet lack security. For example, an office printer is the kind of product most of us purchase and use without giving a minute of thought to security. The software in the printer may be bug-free, but is it able to prevent a would-be eavesdropper from capturing a remote electronic copy of everything you print, including your sensitive financial documents?

Secure systems are robust in the face of persistent attack. Secure systems are able to keep hackers out by design. One key strategy for building secure systems is to validate all inputs, especially those arriving over an open network connection. For example, security could be added to a printer by ensuring against buffer overflows and encrypting and digitally signing firmware updates.

One of the unfortunate facts of designing secure embedded systems is that the hackers who want to get in only need to find and exploit a single weakness. Adding layers of security is good, but if even any one of those layers remains fundamentally weak, a sufficiently motivated attacker will eventually find and breach that defense. But that's not an excuse for not trying.

For years, the largest printer maker in the world apparently gave little thought to the security of the firmware in its home/office printers, even as it was putting tens of millions of tempting targets out into the world. Now the security of those printers has been breached by security researchers with a reasonable awareness of embedded systems design. Said one of the lead researchers, "We can actually modify the firmware of the printer as part of a legitimate document. It renders correctly, and at the end of the job there's a firmware update. ... In a super-secure environment where there's a firewall and no access—the government, Wall Street—you could send a résumé to print out."

Security is a brave new world for many embedded systems designers. For decades we have relied on the fact that the microcontrollers and flash memory and real-time operating systems and other less mainstream technologies we use will protect our products from attack. Or that we can gain enough "security by obscurity" by keeping our communications protocols and firmware upgrade processes secret. But we no longer live in that world. You must adapt.

Consider the implications of an insecure design of an automotive safety system that is connected to another Internet-connected computer in the car via CAN; or the insecure design of an implanted medical device; or the insecure design of your product.

Security is a brave new world for many embedded systems designers. For decades, we relied on the technologies we use to protect our products or that we can gain enough "security by obscurity." But we no longer live in that world.

Too often, the ability to upgrade a product's firmware in the field is the very vector that's used to attack. This can happen even when a primary purpose for including remote firmware updates is motivated by security. For example, as I've learned in my work as an expert witness in numerous cases involving reverse engineering of the techniques and technology of satellite television piracy, much of that piracy has been empowered by the same software patching mechanism that allowed the broadcasters to perform security upgrades and electronic countermeasures. Ironically, had the security smart cards in those set-top boxes had only masked ROM images the overall system security may have been higher. This was certainly not what the designers of the system had in mind. But security is also an arms race.

Like reliability, security must be designed in from the start. Security can't be patched or tested or debugged in. You simply can't add security as effectively once the product ships. For example, an attacker who wished to exploit a current weakness in your office printer or smart card might download his hack software into your device and write-protect his sectors of the flash today so that his code could remain resident even as you applied security patches.

RELIABLE AND SECURE EMBEDDED SYSTEMS

It is important to note at this point that reliable systems are inherently more secure. And that, vice versa, secure systems are inherently more reliable. So, although, design for reliability and design for security will often individually yield different results—there is also an overlap between them.

An investment in reliability, for example, generally pays off in security. Why? Well, because a more reliable system is more robust in its handling of all errors, whether they are accidental or intentional. An anti-lock braking system with

Reliable systems are inherently more secure. And vice versa, secure systems are inherently more reliable. So, although, design for reliability and design for security will often individually yield different results—there is overlap between them.

a fall back to mechanical braking for increased reliability is also more secure against an attack against that critical hardware input sensor. Similarly, those printers wouldn't be at risk of fuser-induced fire in the case of a security breach if they were never at risk of fire in the case of any misbehavior of the software.

Consider, importantly, that one of the first things a hacker intent on breaching the security of your embedded device might do is to perform a (mental at least) fault-tree analysis of your system. This attacker would then target his time, talents, and other resources at one or more single points of failure he considers most likely to fail in a useful way.

Because a fault-tree analysis starts from the general goal and works inward deductively toward the identification of one or more choke points that might produce the desired erroneous outcome, attention paid to increasing reliability such as via FMEA usually reduces choke points and makes the attackers job considerably more difficult. Where security can break down even in a reliable system is where the possibility of an attacker's intentionally-induced failure is ignored in the FMEA weighting and thus possible layers of protection are omitted.

Similarly, an investment in security may pay off in greater reliability—even without a directed focus on reliability. For example, if you secure your firmware upgrade process to accept only encrypted and digitally-signed binary images you'll be adding a layer of protection against a inadvertently corrupted binary causing an accidental error and product failure. Anything you do to improve the reliability of communications (through checksums, prevention of buffer overflows, and so forth) can have a similar effect on reliability.

THE ONLY WAY FORWARD

Each year it becomes increasingly important for all of us in the embedded systems design community to learn to design reliable and secure products. If you don't, it might be your product making the wrong kind of headlines and your source code and design documents being pored over by lawyers. It is no longer acceptable to stick your head in the sand on these issues. ■

Probes and analyzers

Excellent article *("Troubleshooting real-time software issues using a logic analyzer," David B. Stewart, March 2012, p. 19, www.eetimes.com/4236800)*. With a bit of extra programming, you can also sometimes use a "companion" microcontroller instead of a dedicated logic analyzer.

—vapats

Learn today. Design tomorrow.
ESD
EMBEDDED SYSTEMS DESIGN
VERIFYING REQUIREMENTS AND OBJECT CODE 12
Discriminated unions 9
Debug real-time SW with logic analyzer 19
Managing multiple processes in multicore 27
Probing pointers 34

I'm wanting to try this out for the fun of it with a pinball machine's logic board.

—K1200LT Rider

Probe pointers

Very informative *("Probing pointers," Jack Ganssle, March 2012, p. 34, www.eetimes.com/4236927)*. I have had $15 "all purpose" probes come back and bite me.

I'm assuming that for emergency run repairs on printed-circuit boards it would be better to arch the patch wire off the PCB than have it lay on top of the board.

—Saluki_456

Discriminated unions

With virtual functions and proper instantiation, you do not need any isKindOf() operator to discriminate which parameters or functions to use *("Discriminated unions," Dan Saks, March 2012, p. 9, www.eetimes.com/4237055)*. This is one of the most powerful underpinnings of the C++ language.

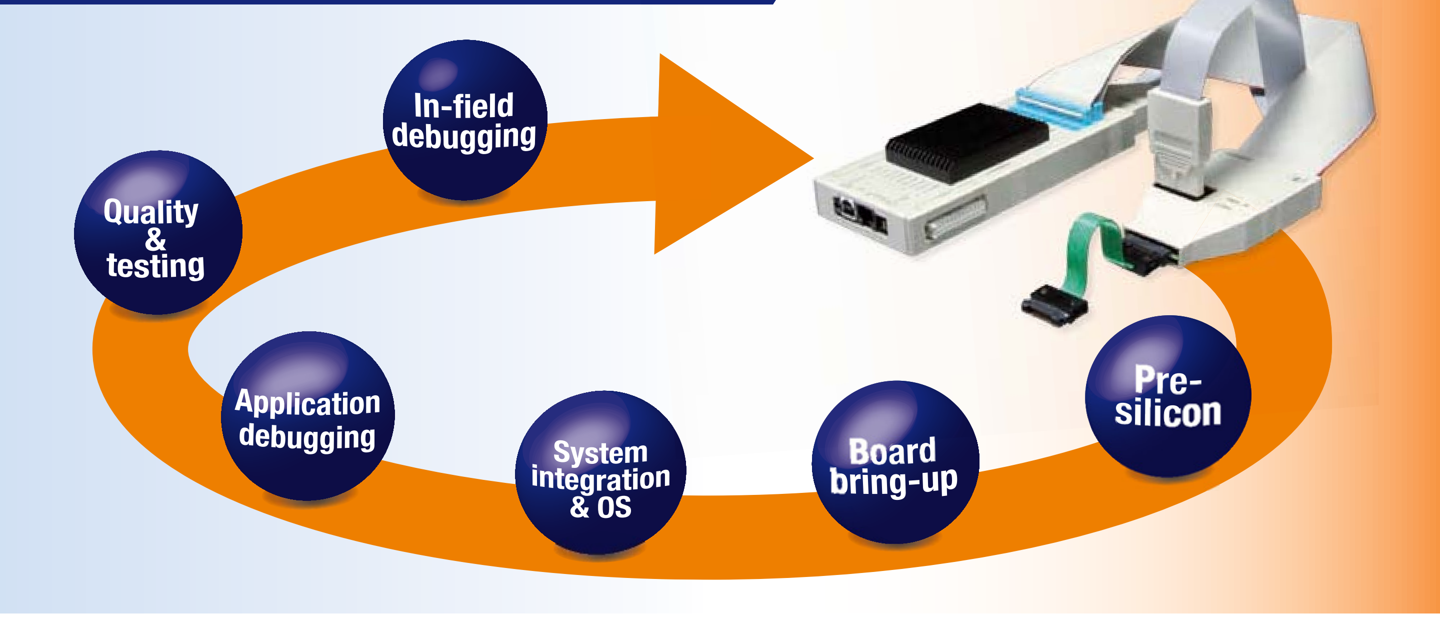
— krwada

We welcome your feedback. Letters to the editor may be edited. Send your comments to Colin Holland at colin.holland@ubm.com or post your feedback directly online, under the article you wish to discuss.

NEWS 2012

LAUTERBACH DEVELOPMENT TOOLS

DEBUGGER, REAL-TIME TRACE, LOGIC ANALYZER



A Debugger for all Phases of the Project

Embedded designs are becoming ever more complex and time to market is getting shorter. To meet these challenges many project managers now rely on debug and trace tools that can accompany developers through all phases of the project.

TRACE32, the debug and trace tool family from Lauterbach provides a consistent concept and environment which can be further extended with user customizable scripts. This helps to shorten the familiarization process and makes time for the actual development work. Developers with practical knowledge gained from more than 10 years experience with TRACE32 are quite common. So, what makes TRACE32 different?

- Hardware- and software-based tools
- Early support for new processors
- Large portfolio of supported processors
- Extensive test and analysis functions
- Seamless integration into the embedded tool chain

Hardware and Software Tools

The core business of Lauterbach is the design and manufacture of hardware-based debug and trace tools. In addition Lauterbach has also offered logic analyzers for over 20 years. The key feature of TRACE32 logic analyzers is seamless integration within the hardware-based debug and trace tools. For a typical application using the logic analyzer integrated in PowerTrace II read the article “Checking JTAG Signals” on page 6.

Fast, efficient computers mean that more simulation and validation is being undertaken on PC and workstations. In the embedded world the pre-silicon software development on virtual targets has become the norm. For this phase of the project Lauterbach can provide pure software solutions.

»

CONTENTS

Virtual Targets

Today virtual targets are increasingly being used to start software development long before the first hardware prototypes become available. As soon as a virtual target is available, debugging of the driver, the operating system, and the application can begin.

For debugging and tracing, most virtual targets have their own API. If this is not the case, the standardized MCD-API (**http://www.lauterbach.com/mcd_api.html**) can be used. Many new projects today use multicore chips. Consequently, Lauterbach has expanded its multicore debugging support for virtual targets since 2011.

Pre-Silicon Validation

For semiconductor manufacturers, it is important to validate the design of their processors or SoCs before actual production. Individual sections are intensely tested, for example: the JTAG interface, the entire core, or the interaction between core and peripherals.

For this testing, you traditionally used an emulator for the silicon (e.g. Palladium) or FPGA prototypes, connected to the hardware-based TRACE32 debug tools. This would run much slower than the real processors.

Today, you can perform first validations of Verilog or SystemC models directly on a PC or a workstation. With pure software validation you cannot use debug hardware. Therefore, Lauterbach added a Verilog Back-End to its software in 2011. This simulates a JTAG interface at the signal level (see figure 1).

The integration of TRACE32 tools into the pre-silicon validation forms an important part of the early support for the latest processors and SoCs:

- Tested tools are ready before the first silicon leaves the factory.
- Expert knowledge of the new processor/SoC is available and can be accessed by the customer.
- Start-up scripts for the TRACE32 debugger are available.

60+ Supported Processor Architectures

Lauterbach has tools available for all the common processors or SoCs on the embedded market. In fact Lauterbach is the only provider of tools for many cores. Standard controllers, DSPs, FPGA softcores, configurable cores - everything can be combined into a multicore chip and debugged with a TRACE32 tool.

In 2011, Lauterbach also added support for numerous new processors and multicore chips. For an overview, see the table on page 4.

Test and Analysis Functions

Each phase of a project requires its own test and analysis functions. To provide this, the TRACE32 PowerView GUI includes an extensive selection of commands and menus. Boundary scan commands (see figure 2), core detection commands and commands for manipulating the JTAG pins are some examples of low-level commands.

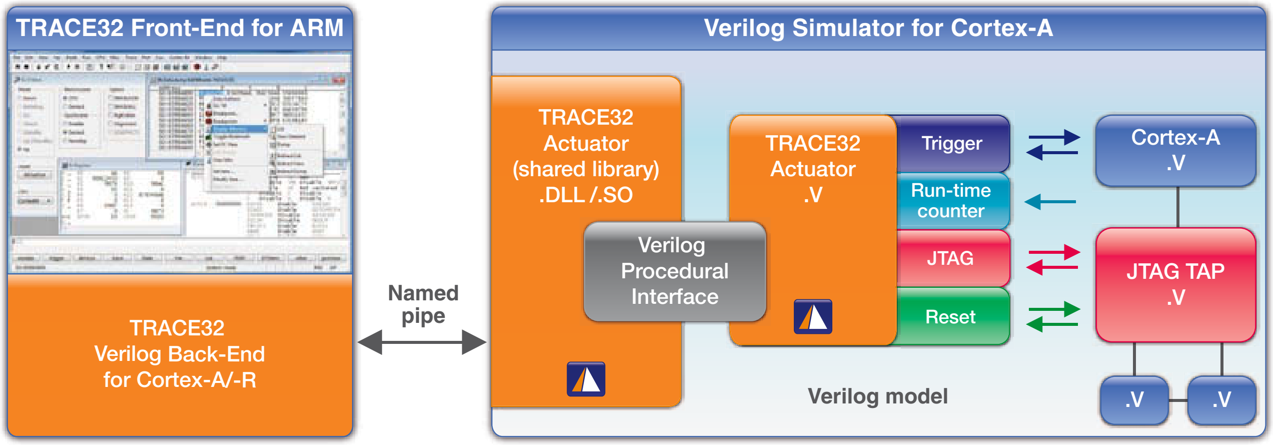
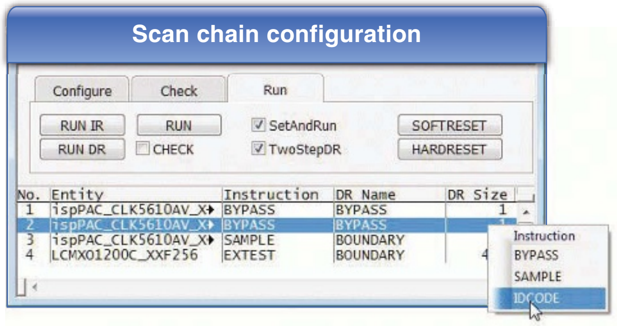

TRACE32 Front-End for ARM
TRACE32 Verilog Back-End for Cortex-A/-R
Named pipe
Verilog Simulator for Cortex-A
TRACE32 Actuator (shared library) .DLL /.SO
Verilog Procedural Interface
TRACE32 Actuator .V
Trigger
Run-time counter
JTAG
Reset
Cortex-A .V
JTAG TAP .V
.V
.V
Verilog model

Fig. 1: For each user entry in TRACE32 Front-End, Verilog Back-End produces JTAG signals for validation of the model.


Scan chain configuration

No.	Entity	Instruction	DR Name	DR Size
1	ispPAC_CLK5610AV_X	BYPASS	BYPASS	1
2	ispPAC_CLK5610AV_X	BYPASS	BYPASS	1
3	ispPAC_CLK5610AV_X	SAMPLE	BOUNDARY	
4	LCMXO1200C_XXF256	EXTEST	BOUNDARY	4

Fig. 2: Boundary-Scan commands are available for commissioning the hardware.

During the quality and test phase the high-level commands provide support for the developer and these typically deal with analysis of trace data. Examples are: measuring function runtime, energy profiling, or details of code-coverage.

Since the beginning of 2011, Lauterbach has enabled most major processor architectures to stream the trace information to the host computer in real-time. This allows significantly more diagnostic data to be collected and quality assurance becomes much easier. For more information, see the article “Code-Coverage – Simplified” on page 7.

Integration into Embedded Tool Chain

The TRACE32 software is an open design so that it works smoothly with all of the common basic components of an embedded design. This includes:

- Host operating systems
- Programming languages and compilers
- Target operating systems
- Virtual machines, such as Android VM Dalvik

The open TRACE32 API allows seamless interaction with numerous third-party tools. Examples include special IDEs such as Eclipse, graphical programming tools and external profiling tools. Several new developments in this area were added in 2011.

Prism, the parallelization tool from the Scottish company CriticalBlue, supports developers when migrating single-core code to run on multicore chips. The tool enables you to try different parallelization strategies without making changes to the function code. When the optimal strategy is determined, the parallelization can be performed step by step, also supported by Prism.

Since July 2011, Lauterbach has included the option of exporting trace information in Prism format, enabling the CriticalBlue tools to work with the trace recorded by the actual operation of the code.

The article “Simulation and Reality Come Closer Together” on page 14 thoroughly describes another innovation – the integration between MATLAB Simulink® and TRACE32.

Extended Lifetime

When migrating to a new technology Lauterbach has a philosophy of ensuring there is a long transition phase. They will not force a customer to accept a technology change while in the middle of a key project.

For example: Starting in May 2012, Lauterbach will introduce a QT version of its graphical user interface TRACE32 PowerView (see figure 3). With QT, an up-to-date GUI will be available for Linux, Mac OS X, and other host operating systems.

Lauterbach will continue to support the Motif version of TRACE32 PowerView so that customers can determine their own best transition time.

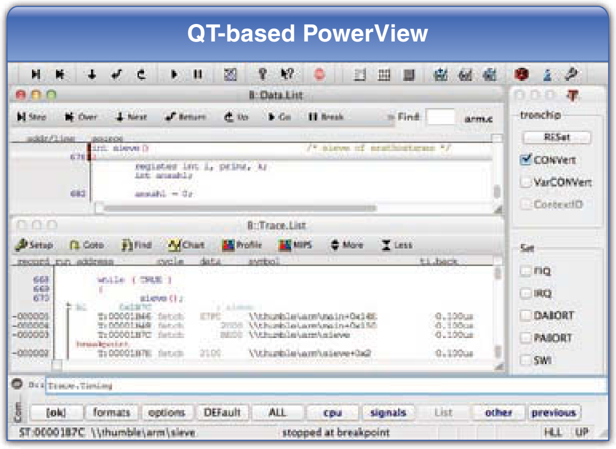

QT-based PowerView

Fig. 3: The new QT-based GUI for Linux, Mac OS X and other operating systems.

Within these pages of our NEWS 2012, you will find further information which might be useful for your current or future projects. Hopefully you will find a feature that contributes to your project’s success. We will be demonstrating several of them live at the upcoming **ESC Silicon Valley, March 26-29th, in San Jose**, and at many other shows in the US throughout the year.

New Supported Processors

New Derivatives	
Altera	**Cortex-A/-R** • FPGA with Cortex-A9 MPCore as Hardcore **MIPS32** • MP32
AppliedMicro	**PPC44x** • 86290/491/791 Q2/2012
ARM	**Cortex-A/-R** • Cortex-A7/Cortex-A7 MPCore • Cortex-A15 • Cortex-A15 MPCore • Cortex-R5/Cortex-R5 MPCore • Cortex-R7/Cortex-R7 MPCore
Beyond Semiconductor	**Beyond** • BA22
Broadcom	**MIPS32** • BCM35230 • BCM63168, BCM63268 • BCM7231, BCM7358
Cavium	**MIPS64** • CN61XX/CN62XX/CN66XX • CN67XX/CN68XX
Ceva	**CEVA-X** • CEVA-XC
CSR	**ARM11** • QUATRO 4500
Cypress	**ARM9** • EZ-USB FX3
Energy Micro	**Cortex-M** • Giant Gecko
Freescale	**MCS12X** • MC9S12VR, MC9S12XS • MM912F634 **Cortex-A/-R** • i.MX 6 Series **MPC55xx/56xx** • MPC5604E, MPC5675K, • MPC5676R **Power QUICC III** • P1010, P1020 • P2040, P2041 • P3041, P4040, P4080 • PSC9131 **QorIQ 64-Bit** • P5010, P5020
Fujitsu	**Cortex-A/-R** • MB9DF126, MB9EF126
IBM	**PPC44x** • 476FP Q2/2012
Ikanos	**MIPS32** • Fusiv Vx185
Infineon	**TriCore** • TriCore Multi-Core Architecture
Intel®	**Atom™/x86** • Atom D2500, Atom N550 • Core i3/i5/i7 2nd Generation
Lantiq	**MIPS32** • XWAY xRX100 • XWAY xRX200
LSI	**PPC44x** • ACP344x Q2/2012
Marvell	**ARM9 Debug-Cabel** • 88E7251 **ARM11 Debug-Cabel** • 88AP610-V6, MV78460-V6 **Cortex-A/-R Debug-Cabel** • 88AP610-V7, MV78460-V7
Nuvoton	**Cortex-M** • NuMicro
NXP	**Cortex-M** • LPC12xx **Beyond** • JN5148
Qualcomm	**MIPS32** • AR7242 **Cortex-A/-R** • Krait
Renesas	**V850** • V850E2/Fx4: 70F3548..66 70F4000..70F4011 • V850E2/Fx4-L: 70F3570..89 • V850E2/Px4: 70F3503/05 70F3507/08/09 **78K0R/RL78** • 78K0R/Kx3-C/L • RL78/G14, RL78/G1A • RL78/F12, RL78/I1A **SH** • SH708x with AUD/Onchip-Trace • SH7147

New Derivatives	
Samsung	**ARM7** • S3F4 **Cortex-A/-R** • S5PV310 **Cortex-M** • S3FM, S3FN
ST-Ericsson	**Cortex-A/-R** • A9500, A9540, M7400 **MMDSP** • A9500, A9540
STMicro-electronics	**MPC55xx/56xx** • SPC56A80, SPC56HK **Cortex-M** • STM32F2xx, STM32F4xx
Synopsys	**ARC** • ARC EM4, ARC EM6
Tensilica	**Xtensa** • BSP3, LX4, SSP16
Texas Instruments	**MSP430** • CC430Fxxx, MSP430FR5xxx • MSP430x1xx..MSP430x6xx
Texas Instruments (Cont.)	**ARM9** • AM38xx • OMAP4460/4470 • TMS320C6A81xx • TMS320DM81xx **Cortex-A/-R** • AM335x, AM38xx • OMAP4460/4470/543x • RM48L950 • TMS320C6A81xx • TMS320DM81xx • TMS570LS3xxx **Cortex-M** • AM335x • OMAP4460/4470/543x • TMS470MFxxx **TMS320C28X** • TMS320C28346/F28069 **TMS320C6x00** • OMAP4460/4470/543x • TMS320C6A81xx • TMS320DM81xx • TMS320TCI6616/18
Xilinx	**Cortex-A/-R** • Zynq7000

Nexus-Trace Also for Small Package Format Cores

The Nexus cell, which is integrated into the controllers of the MPC560xB/C family from Freescale or the SPC560B/C controllers of ST, can generate trace data for the instructions executed by the core. If an operating system is used, information on task switching are produced as well.

A microcontroller must have a trace interface, so that an external trace tool, such as TRACE32, can record this trace data. However, the members of the MPC560xB/C family do not have this interface in their standard packaging. To provide access to this valuable data about the program run during the development phase, silicon-compatible microcontrollers in a 208-pin BGA development package are offered, which have a Nexus interface with 4 MDO (Message Data Out) pins.

Since mid-2011, Lauterbach has provided MPC560xB/C adapters that can replace the original controller on the target hardware with a 208-pin controller with Nexus interface.

The MPC560xB/C adapter consists of a suitable MPC560xB/C controller in 208-pin BGA development package and Mictor plug with Nexus interface for connecting TRACE32 trace tools (shown in figure 4 in blue). In addition, a socket adapter from the Tokyo Eletech company is required.

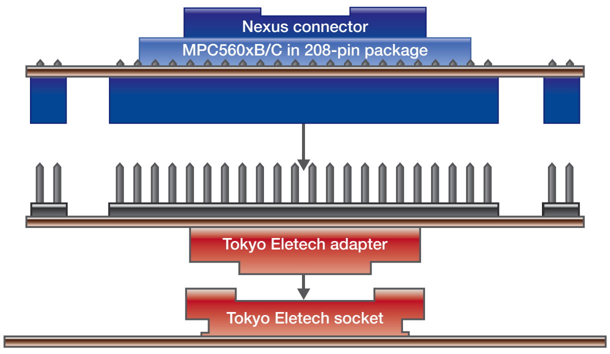


Fig. 4: The MPC560xB/C adapter allows a development package with Nexus interface to be used instead of the original controller.

Checking JTAG Signals

Lauterbach's PowerTrace II is equipped with an integrated logic analyzer and supplied with a standard digital probe. This enables 17 digital channels to be recorded with a sampling rate of up to 200 MHz. This logic analyzer has a save depth of up to 1024K samples and an example of its use would be the test of the JTAG signals during pre-silicon validation (see figures 6 and 7).

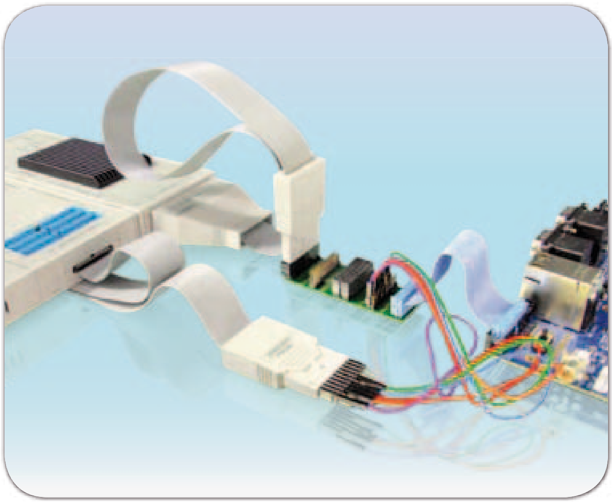


Fig. 5: Measuring arrangement for recording the JTAG signals.

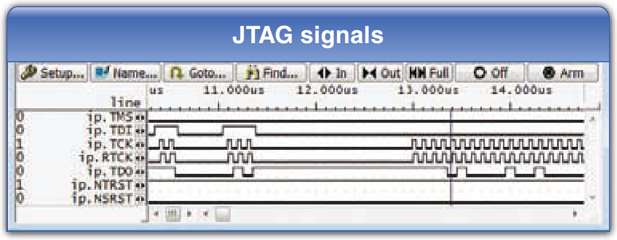
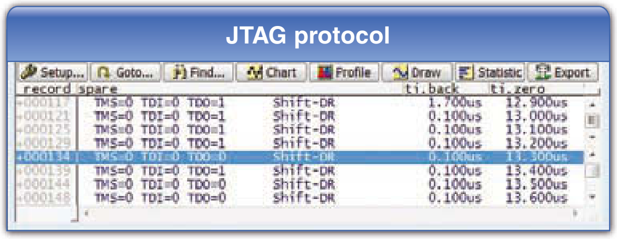


Fig. 6: The recorded JTAG signals.



```
record spare                                        ti.back    ti.zero
-000117   TMS=0 TDI=0 TDO=1       Shift-DR          1.700us    12.900us
-000121   TMS=0 TDI=0 TDO=1       Shift-DR          0.100us    13.000us
-000125   TMS=0 TDI=0 TDO=1       Shift-DR          0.100us    13.100us
-000129   TMS=0 TDI=0 TDO=1       Shift-DR          0.100us    13.200us
+000134   TMS=0 TDI=0 TDO=0       Shift-DR          0.100us    13.300us
-000139   TMS=0 TDI=0 TDO=1       Shift-DR          0.100us    13.400us
-000144   TMS=0 TDI=0 TDO=0       Shift-DR          0.100us    13.500us
-000148   TMS=0 TDI=0 TDO=0       Shift-DR          0.100us    13.600us
```

Fig. 7: The protocol representation of the JTAG signals.

Enhancements to Target OS-Awareness

The following version adaptations have been made:

- eCos 3.0
- Linux v3.0
- SMX v4
- embOS 3.80
- MQX 3.6
- FreeRTOS v7
- RTEMS 4.10

- The content of the QNX tracelogger can be displayed using TRACE32 QNX OS-Awareness. A graphical representation of the task switch is also possible using the TRACE32 command group LOGGER.Chart.
- TRACE32 QNX OS-Awareness has been adapted for the use of position-independent executables.

New Supported Target-OS	
µC/OS-II for Andes	available
Elektrobit tresos (OSEK/ORTI)	available
Erika (OSEK/ORTI)	available
FreeRTOS für AVR32	available
Linux for Beyond	planned
MQX for ARC	available
OSEK/ORTI SMP	planned
PikeOS	available
PXROS-HR Run Mode Debugging	available
RTEMS for Nios II	available
Sciopta 2.x	available
SYS/BIOS for ARM	available
VxWorks SMP	available

Code-Coverage – Simplified

As of March 2011, TRACE32 trace information can be streamed to a host hard-disk from the running target. The large amount of program flow data which can result from this method, leads to a significant simplification of the code-coverage.

Trace-based Code-Coverage

Proof of statement coverage and condition coverage is often required to meet system quality specifications in industries such as medical and automotive.

- **Statement coverage** proves that each line of code was executed during the system test.
- **Condition coverage** proves that for each conditional instruction both pass and fail branches were executed at least once.

For many embedded systems highly optimized code must be tested in real-time. The alternatives of code instrumentation and non-real-time operation cannot be used in these cases.

To be able to meet these requirements, the target processor/SoC must fulfill the following prerequisites:

1. The cores which are implemented must have a core trace logic (see figure 8). This logic generates information about the instructions executed by the core. Depending on the operation of the trace logic, information about the task switches and the read/write operations can also appear.

2. The processor/SoC must have a trace port with sufficient bandwidth so that the trace information can be recorded by an external tool without any information loss.

The Classic Measurement Process

Until now, code-coverage analysis was performed with TRACE32 using the following steps:

1. Start program execution and automatically stop when the trace memory is full.
2. Transfer the trace memory content to the code-coverage database.
3. Continue program execution.

For each measurement step, the amount of data collected was limited by the size of the memory available within the trace tool. The results of the code-coverage-analysis could be checked after the total measurement was completed or, if needed, after each intermediate step.

New: Streaming

If the trace data is transferred to a drive on the host computer at the time of recording, the complete software routine can be recorded **in one measurement step**. The streamed data is stored within a file on the hard-disk. To avoid completely filling the hard-disk with trace data, TRACE32 stops streaming as soon as less than 1 GByte of free memory remains.

To be able to stream, the following technical prerequisites must be fulfilled:

- 64-bit host computer and 64-bit TRACE32 executable
- Interface between trace tool and host computer must be as fast as possible.
- Optimal configuration of the trace source and the trace tool

»

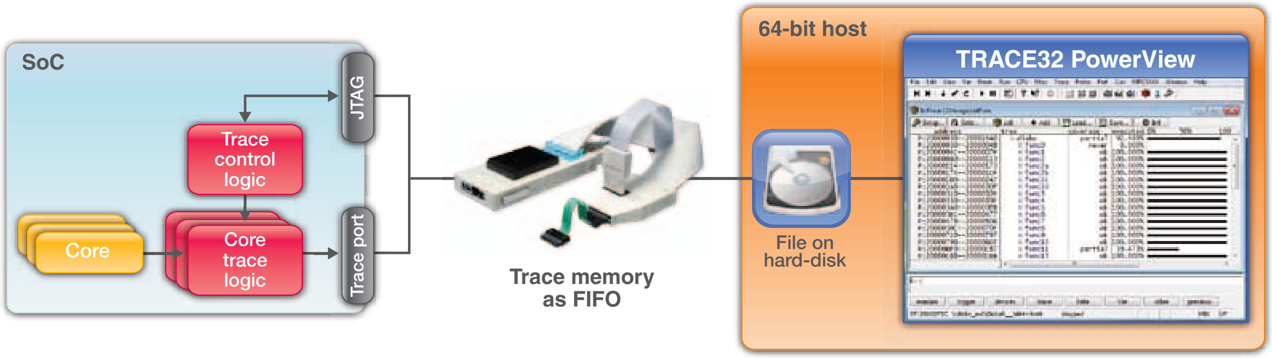


Fig. 8: For the code-coverage analysis, up to 1 TByte of trace data can be streamed to the host computer.

Fast Host Interface

The amount of trace data that is exported via the trace port depends on the target system hardware. The number of cores, the number of trace port pins, and the trace clock speed are all important parameters. The protocol used by the core trace logic plays also an important role. For example, the ARM PTM protocol is more compact than the ARM ETMv3 protocol (see figure 9).

The embedded software is another major variable. A software program that performs many jumps and retrieves data/instructions mainly from the cache produces more trace data per second than a software program that processes many sequential instructions and must frequently wait for the availability of data/instructions.

The amount of data varies but it is always large. Streaming only works properly, if the transfer rate between the tool and the host computer is fast enough to transfer all of the data from the trace port to the host computer without any data loss. The 1 GBit Ethernet interface is the only recommended interface for the PowerTrace II.

The programming of the trace logic on the chip can be used to directly influence the amount of trace data being generated. The logic should be programmed so that only trace information which is relevant to the code-coverage analysis is being generated. To illustrate this point, the following two examples are provided.

ETM/PTM: Optimal Configuration

ETM and PTM are different implementations of the core trace logic on the ARM/Cortex architectures. The ETM can be configured so that trace information is produced only for the instructions executed by the program. Information about the read/write operations is not needed for code-coverage. By default the PTM only generates information about the program flow. Therefore the PTM does not need to be configured.

Both trace sources encode the virtual address instructions. If an embedded design uses an operating system, such as Linux or Embedded Windows, virtual addresses cannot be mapped unambiguously to physical

PowerTrace vs. PowerTrace II

TRACE32 trace tools are available in two designs, which differ especially in relation to their features.

PowerTrace

- 256 or 512 MByte trace memory
- USB 2.x and 100 MBit Ethernet
- 80 MBit/s as maximum transfer rate to host computer
- Software compression of trace data (factor 3)
- Memory interface with 100 MHz

PowerTrace II

- 1/2/4 GByte trace memory
- USB 2.x and 1 GBit Ethernet
- 500 MBit/s as maximum transfer rate to host computer
- Hardware compression of trace data for ETMv3 and PTM (factor 6)
- Memory interface with 233 MHz

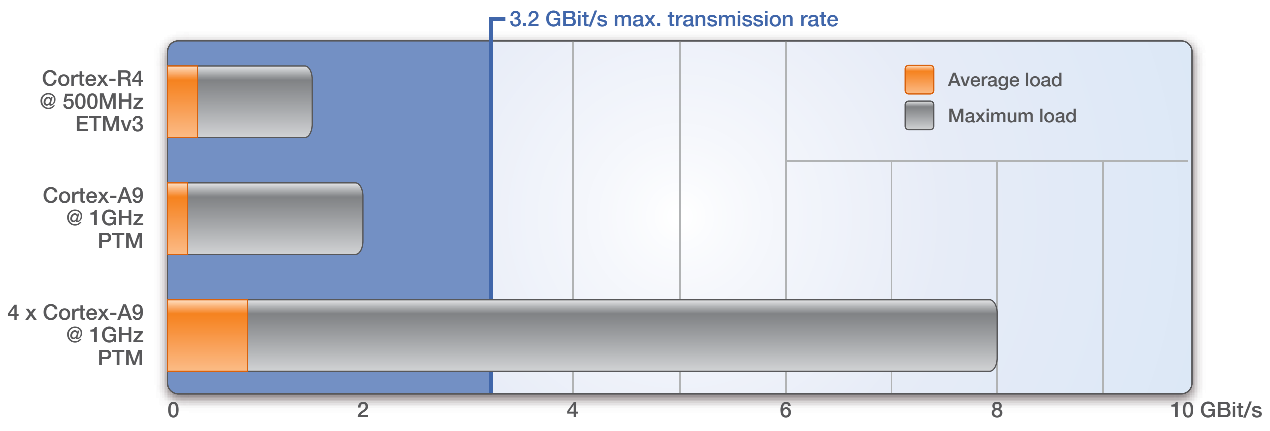


Fig. 9: A transmission rate of 3.2 GB/s is generally adequate for streaming program sequence information on the host.

addresses. The trace source must also be configured, so that information is generated defining the virtual address space in which an instruction was located.

For the ARM ETM/PTM, the amount of trace data can be further reduced:

- The code-coverage analysis does not analyze or need time information. We therefore recommend configuring the TRACE32 trace tool so that the trace data is transferred to the host without time stamps. This reduces the amount of data by a third.

- PowerTrace II also provides FPGA-based hardware compression of the trace data. This enables up to 3.2 GBit/s trace data to be transferred to the host computer. Figure 9 shows that this transfer rate is generally sufficient for streaming ETM/PTM data without any data loss.

Nexus: Optimal Configuration

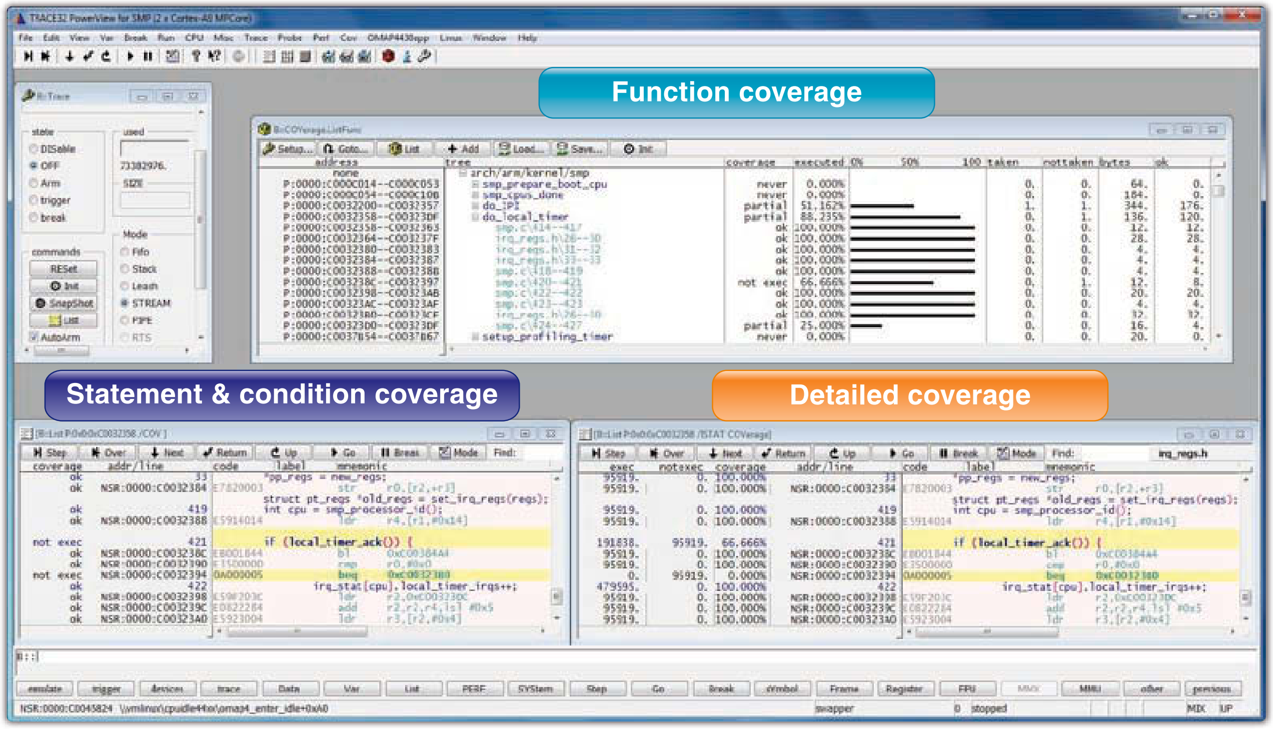
On processors of the MPC5xxx/SPC5xx families the core trace logic is implemented to the Nexus standard. To undertake code-coverage analysis, a Nexus class 2 trace cell is adequate as all you need is detail of the program sequence on the individual core(s). If Branch History Messaging is used this can make the trace data very compact. Compared to standard trace data a reduction by a factor of 10 is realistic. Only PowerTrace II supports streaming from the Nexus trace port.

Streaming also works for all other processors/SoCs that are supported by TRACE32 and have a trace port.

Code-Coverage for SMP-Systems

TRACE32 also supports code-coverage analysis on SMP (symmetric multiprocessing) systems. For code-coverage it must be proven that an instruction was executed, which core was responsible for running the code is irrelevant. Figure 10 shows the results of code-coverage for two Cortex-A9 MPCores.

For statement and condition coverage, if only the fail-branch of a conditional statement was run the statement is highlighted in yellow and marked with "not exec". The detailed coverage lists the specifics of how often each statement or each branch of the statement was run.


Function coverage
Statement & condition coverage
Detailed coverage
TRACE32 PowerView for SMP (2 x Cortex-A9 MPCore)
File Edit View Var Break Run CPU Misc Trace Probe Perf Cov OMAP4430app Linux Window Help
B::Trace
state
DISable
OFF
Arm
trigger
break
used
73382976.
SIZE
commands
RESet
Init
SnapShot
List
AutoArm
Mode
Fifo
Stack
Leash
STREAM
PIPE
RTS
B::COVerage.ListFunc
Setup... Goto... List Add Load... Save... Init
Step Over Next Return Up Go Break Mode Find:
irq_regs.h
emulate trigger devices trace Data Var List PERF SYStem Step Go Break sYmbol Frame Register FPU MMX MMU other previous
swapper
stopped
MIX UP

```
address                     tree                          coverage  executed  taken  nottaken  bytes  ok
none                        arch/arm/kernel/smp
P:0000:C000C014--C000C053   smp_prepare_boot_cpu          never       0.000%   0.     0.      64.     0.
P:0000:C000C054--C000C108   smp_cpus_done                 never       0.000%   0.     0.     184.     0.
P:0000:C0032200--C0032357   do_IPI                        partial    51.162%   1.     1.     344.   176.
P:0000:C0032358--C00323DF   do_local_timer                partial    88.235%   0.     1.     136.   120.
P:0000:C0032358--C0032363     smp.c\414--417              ok        100.000%   0.     0.      12.    12.
P:0000:C0032364--C003237F     irq_regs.h\26--30           ok        100.000%   0.     0.      28.    28.
P:0000:C0032380--C0032383     irq_regs.h\31--32           ok        100.000%   0.     0.       4.     4.
P:0000:C0032384--C0032387     irq_regs.h\33--33           ok        100.000%   0.     0.       4.     4.
P:0000:C0032388--C003238B     smp.c\418--419              ok        100.000%   0.     0.       4.     4.
P:0000:C003238C--C0032397     smp.c\420--421              not exec   66.666%   0.     1.      12.     8.
P:0000:C0032398--C00323AB     smp.c\422--422              ok        100.000%   0.     0.      20.    20.
P:0000:C00323AC--C00323AF     smp.c\423--423              ok        100.000%   0.     0.       4.     4.
P:0000:C00323B0--C00323CF     irq_regs.h\26--30           ok        100.000%   0.     0.      32.    32.
P:0000:C00323D0--C00323DF     smp.c\424--427              partial    25.000%   0.     0.      16.     4.
P:0000:C0037B54--C0037B67   setup_profiling_timer         never       0.000%   0.     0.      20.     0.
```

```
[B::List P:0x0:0xC0032358 /COV]
coverage   addr/line              code      label  mnemonic
      ok                     33                    *pp_regs = new_regs;
      ok   NSR:0000:C0032384  E7820003             str     r0,[r2,+r3]
                                                   struct pt_regs *old_regs = set_irq_regs(regs);
      ok                    419                    int cpu = smp_processor_id();
      ok   NSR:0000:C0032388  E5914014             ldr     r4,[r1,#0x14]

not exec                    421                    if (local_timer_ack()) {
      ok   NSR:0000:C003238C  EB001844             bl      0xC00384A4
      ok   NSR:0000:C0032390  E3500000             cmp     r0,#0x0
not exec   NSR:0000:C0032394  0A000005             beq     0xC00323B0
      ok                    422                       irq_stat[cpu].local_timer_irqs++;
      ok   NSR:0000:C0032398  E59F203C             ldr     r2,0xC00323DC
      ok   NSR:0000:C003239C  E0822284             add     r2,r2,r4,lsl #0x5
      ok   NSR:0000:C00323A0  E5923004             ldr     r3,[r2,#0x4]
B::
NSR:0000:C0043824  \\vmlinux\cpuidle44xx\omap4_enter_idle+0xA0
```

```
[B::List P:0x0:0xC0032358 /STAT COVerage]
    exec  notexec  coverage  addr/line              code      label  mnemonic
  95919.       0.  100.000%                    33                    *pp_regs = new_regs;
  95919.       0.  100.000%  NSR:0000:C0032384  E7820003             str     r0,[r2,+r3]
                                                                     struct pt_regs *old_regs = set_irq_regs(regs);
  95919.       0.  100.000%                   419                    int cpu = smp_processor_id();
  95919.       0.  100.000%  NSR:0000:C0032388  E5914014             ldr     r4,[r1,#0x14]

 191838.   95919.   66.666%                   421                    if (local_timer_ack()) {
  95919.       0.  100.000%  NSR:0000:C003238C  EB001844             bl      0xC00384A4
  95919.       0.  100.000%  NSR:0000:C0032390  E3500000             cmp     r0,#0x0
      0.   95919.    0.000%  NSR:0000:C0032394  0A000005             beq     0xC00323B0
 479595.       0.  100.000%                   422                       irq_stat[cpu].local_timer_irqs++;
  95919.       0.  100.000%  NSR:0000:C0032398  E59F203C             ldr     r2,0xC00323DC
  95919.       0.  100.000%  NSR:0000:C003239C  E0822284             add     r2,r2,r4,lsl #0x5
  95919.       0.  100.000%  NSR:0000:C00323A0  E5923004             ldr     r3,[r2,#0x4]
```

Fig. 10: Code-coverage analysis for an SMP system.

CoreSight Trace Memory Controller

The new CoreSight Trace Memory Controller provides SoC designers with more design options for the trace infrastructure. TRACE32 already has support for the first designs which use the TMC.

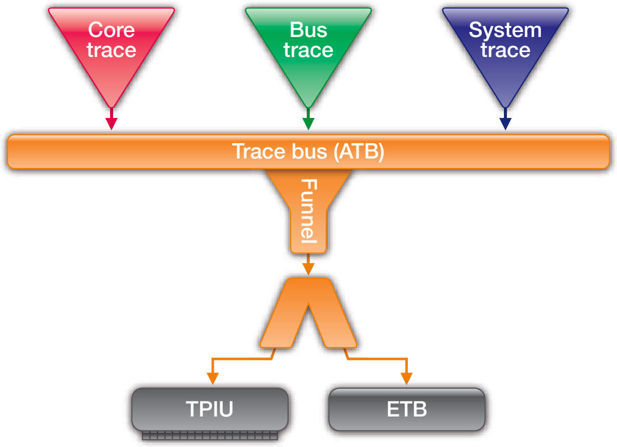


Fig. 11: CoreSight Funnel combines all trace data produced by trace macrocells into a single data stream.

Through CoreSight, the diagnosis data needed for the analysis of SoC-internal processes is produced by 'trace macrocells'. There are three types of trace macrocells:

- **Core trace macrocells** are assigned to a core and generate trace information about the instructions processed by that core. Information about process switches and load/store operations is generated depending on the design of the trace cell.
- **Bus trace macrocells** are firmly assigned to a bus and generate trace information on data transfers that occur on the bus.
- **System trace macrocells** generate trace information for hardware trigger (system event tracing) or provide diagnostic information produced by code instrumentation of the application software.

The CoreSight Funnel combines all of the trace data into a single data stream (see figure 11). This trace data stream is then either stored in an on-chip memory buffer (ETB) or exported to an external tool using a trace port (TPIU). The IP for CoreSight trace being implemented today is sometimes pushed to the limit when dealing with complex multicore SoCs that contain many trace macrocells.

ARM CoreSight

With CoreSight, ARM makes available an extensive set of IP blocks, which enables SoC designers to build a custom debug and trace infrastructure.

A single debug interface is enough to control and coordinate all cores of the SoC, as well as access all memory.

One trace interface is sufficient for providing diagnostic data about the processes occurring within the SoCs without any impact on real-time performance.

- **ETB:** The on-chip trace memory is often too small to record enough trace data for any meaningful future analysis. The typical size for the ETB is still between 4 and 16 KByte.
- **TPIU:** System states may occur where more trace data is being generated than the trace port can output. The CoreSight design is such that trace data from the trace macrocells is only taken over if the trace data can be exported by the TPIU. If the trace data generated remains in the trace macrocells for too long, the FIFOs there can overflow and important data may be lost.

The new CoreSight Trace Memory Controller should provide a solution for both of the above scenarios.

TMC as Embedded Trace Buffer

To be able to store more trace data on-chip for later analysis, the chip manufacturer can theoretically connect up to 4 GByte of SRAM to the Trace Memory Controller (see figure 12).

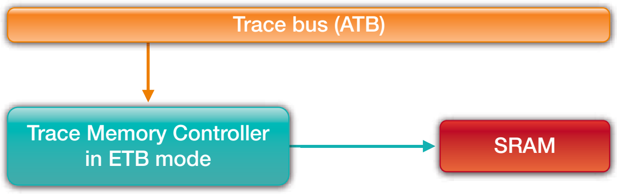


Fig. 12: In ETB mode, the Trace Memory Controller can make up to 4 GByte of on-chip trace memory available.

TMC as Embedded Trace FIFO

Inspections of the trace data streams being exported by the TPIU have shown that the bandwidth of most trace ports is large enough for normal operation. Overload, and therefore loss of trace data, only happens when peaks occur.

The Trace Memory Controller can be integrated into the trace infrastructure of the SoCs, so that the Trace Memory Controller acts as an Embedded Trace FIFO and cushions peaks in the load on the TPIU (see figure 13). This ETF is designed so that no trace data loss can occur. The size of the ETF can be freely defined from 512 Bytes to 4 GBytes.

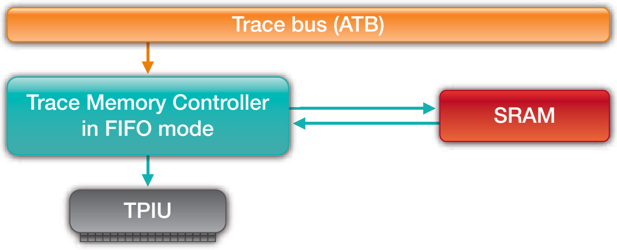


Fig. 13: In FIFO mode, the Trace Memory Controller can cushion load peaks on the TPIU. By doing this, trace data loss can be avoided.

Both integrations of the Trace Memory Controller in the trace infrastructure depicted are simple examples. Of course, you can build the TMC IP block into the CoreSight system in much more complex and flexible ways.

Modifications in TRACE32

As you would expect, Lauterbach has to modify the TRACE32 software for the configuration and handling of the Trace Memory Controller. This applies especially when the Trace Memory Controller is integrated in the SoC using new, previously unsupported ways. The TRACE32 user only needs to configure the basic address for the TMC. Then all the proven trace display and analysis features can be used as usual.

TMC as Router to High-Speed Link

The idea of moving away from dedicated trace ports has long been discussed within the embedded community. There are certainly several good arguments for this move.

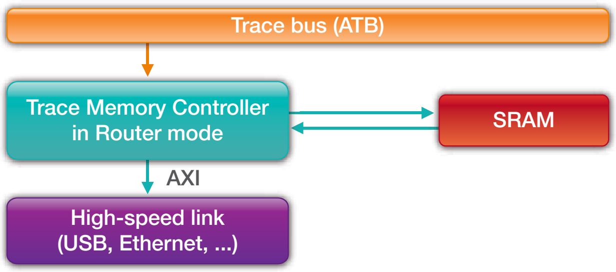

Trace bus (ATB)
Trace Memory Controller in Router mode
SRAM
AXI
High-speed link (USB, Ethernet, ...)

Fig. 14: In Router mode, the Trace Memory Controller forwards the trace data for the export to a high-speed standard interface.

For the first time CoreSight traces can now connect to a high-speed standard interface by using the Trace Memory Controller. USB or Ethernet interfaces are common favorites, especially as they are available in many end products. Ideally, the external trace tool will share the interface with the other connected devices.

Within the SoC, the TMC operates as Embedded Trace Router and has the task of passing on the trace data through the AXI bus for the export to the IP of the high-speed interface (see figure 14).

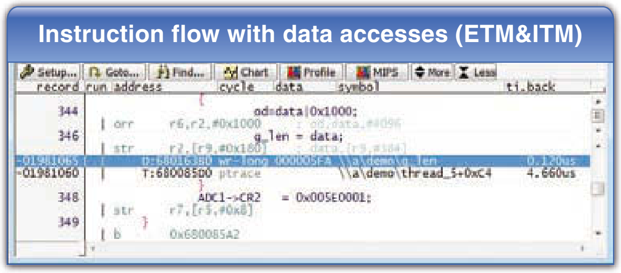
This new method of trace export will need completely new trace tools. Lauterbach is currently in close contact with leading semiconductor manufacturers to develop the appropriate tools for this switch in technology.

TRACE32 CoreSight Features

- Open for use with all cores which can be integrated into CoreSight; Lauterbach offers debug solutions for all ARM/Cortex cores and for numerous DSPs, as well as for configurable cores.
- Support for asymmetric multiprocessing (AMP) and symmetric multiprocessing (SMP)
- Debugging via JTAG interface and 2-pin Serial Wire Debug
- Synchronized debugging of all cores
- Support for the CoreSight Cross Trigger Matrix
- Support for all types of trace macrocells (ETM, PTM, HTM, ITM, STM, and more)
- Tools for parallel and serial trace ports
- Multicore tracing

Intelligent Trace Analyses for Cortex-M3/M4

Troubleshooting, performance tuning and code-coverage - all of these can be performed quickly and precisely on an embedded system if the adequate trace analysis is provided. In 2011, Lauterbach explored new paths to enable optimized trace analyses for the Cortex-M3/M4 processors.



```
 record run address       cycle  data      symbol                     ti.back
                {
    344                      od=data|0x1000;
          | orr     r6,r2,#0x1000      ; od,data,#4096
    346                      g_len = data;
          | str     r2,[r9,#0x180]     ; data,[r9,#384]
-01981065 |     D:68016380 wr-long 000005F4  \\a\demo\g_len               0.120us
-01981060 |     T:680085D0 ptrace           \\a\demo\thread_5+0xC4      4.660us
                }
    348              ADC1->CR2   = 0x005E0001;
          | str     r7,[r5,#0x8]
    349         }
          | b       0x680085A2
```

Fig. 15: By combining ETM and ITM trace data, read/write accesses can be integrated seamlessly into the program sequence.

Combining ETM and ITM

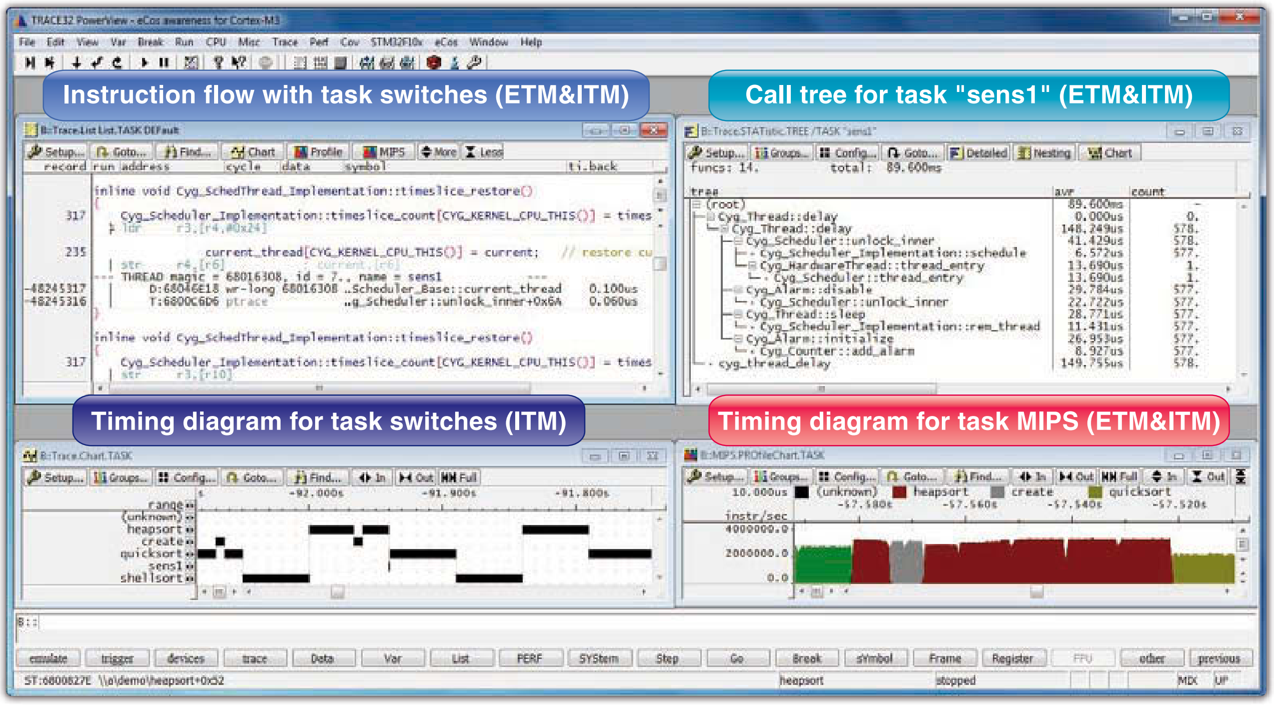
For Cortex-M3/M4 processors, trace information can be generated from two different sources (see figure 17). The **ETMv3** generates information about the executed instructions. The **ITM** generates information about the performed read/write accesses assisted by the Data Watchpoint and Trace Unit (DWT).

The ITM trace packages for read/write accesses contain the following information: data address, data value, program counter.

Through analysis of the program counter, the data accesses which are separately generated can be seamlessly integrated into the program sequence (see figure 15), which in turn leads to significantly simpler error location. The cause of an error such as an incorrect data value being written into an address can be easily found if the write accesses are embedded into the overall program trace.

OS-Aware Tracing

If an operating system is running on the Cortex-M3/M4, task switch information becomes essential for the trace analysis.



```
 record run address       cycle  data      symbol                     ti.back
inline void Cyg_SchedThread_Implementation::timeslice_restore()
{
    317     Cyg_Scheduler_Implementation::timeslice_count[CYG_KERNEL_CPU_THIS()] = times
          | ldr     r3,[r4,#0x24]
    235              current_thread[CYG_KERNEL_CPU_THIS()] = current;    // restore cu
          | str     r4,[r6]                 current,[r6]
      --- THREAD magic = 68016308, id = 7., name = sens1                 ---
-48245317 |     D:68046E18 wr-long 68016308 ..Scheduler_Base::current_thread    0.100us
-48245316 |     T:6800C6D6 ptrace           ..g_Scheduler::unlock_inner+0x6A   0.060us
}
inline void Cyg_SchedThread_Implementation::timeslice_restore()
{
    317     Cyg_Scheduler_Implementation::timeslice_count[CYG_KERNEL_CPU_THIS()] = times
          | str     r3,[r10]
```

```
funcs: 14.            total:   89.600ms

tree                                                       avr          count
⊟ (root)                                                   89.600ms         -
├⊟ Cyg_Thread::delay                                        0.000us        0.
└⊟ Cyg_Thread::delay                                      148.249us      578.
  ├⊟ Cyg_Scheduler::unlock_inner                           41.429us      578.
  │├ · Cyg_Scheduler_Implementation::schedule               6.572us      577.
  │└⊟ Cyg_HardwareThread::thread_entry                     13.690us        1.
  │  └ · Cyg_Scheduler::thread_entry                       13.690us        1.
  ├⊟ Cyg_Alarm::disable                                    29.784us      577.
  │└ · Cyg_Scheduler::unlock_inner                         22.722us      577.
  ├⊟ Cyg_Thread::sleep                                     28.771us      577.
  │└ · Cyg_Scheduler_Implementation::rem_thread            11.431us      577.
  ├⊟ Cyg_Alarm::initialize                                 26.953us      577.
  │└ · Cyg_Counter::add_alarm                               8.927us      577.
└ · cyg_thread_delay                                      149.755us      578.
```

Fig. 16: Through the combination of ETM and ITM trace data, extensive trace analysis can be provided for the eCos operating system.

In order to receive information about task switches the following method can be used: Trace information on the write cycle in which the kernel writes the identifier for the current task on the corresponding OS variable can be generated using the ITM. As described above the write access information can be integrated seamlessly into the program flow trace. This improves the readability of the trace listing (see figure 16). The integration of the task switch into the program sequence also forms the basis for the runtime analyses shown in the figure 16.

Three Recording Modes

To record the trace information generated by the Cortex-M3/M4 processors, Lauterbach supports three modes:

- **FIFO mode:** Storing the information in the 128 MByte memory of the TRACE32 CombiProbe.
- **STREAM mode:** Streaming the information to a hard-disk on the host computer.
- **Real-time Profiling:** The trace information is streamed to the host computer and analyzed during runtime.

For the first two recording modes, the trace information is collected and the trace analysis is undertaken after recording is completed.

Each recording mode has its own features. FIFO is the most commonly used mode. It is quick and usually all that is needed for error location and the runtime analyses.

The ETMv3 implemented on Cortex-M3/M4 processors has neither a trigger nor a trace filter. It is not possible to select for recording only those program segments that are needed for troubleshooting. This can mean trace data might have to be collected for a relatively long period in order to cover the area needed for analysis. In this case the STREAM mode can be the best option. The STREAM mode, however, places high demands on the debug environment:

- The large amount of data that results from streaming requires a 64-bit TRACE32 executable. This is needed to allow the address range for the large number of trace entries that will be collected.
- The transfer rate between CombiProbe and host computer must be fast enough to stream all trace data without a data loss. The 128 MByte memory of the CombiProbe is used to cushion load peaks from the trace port (TPIU).

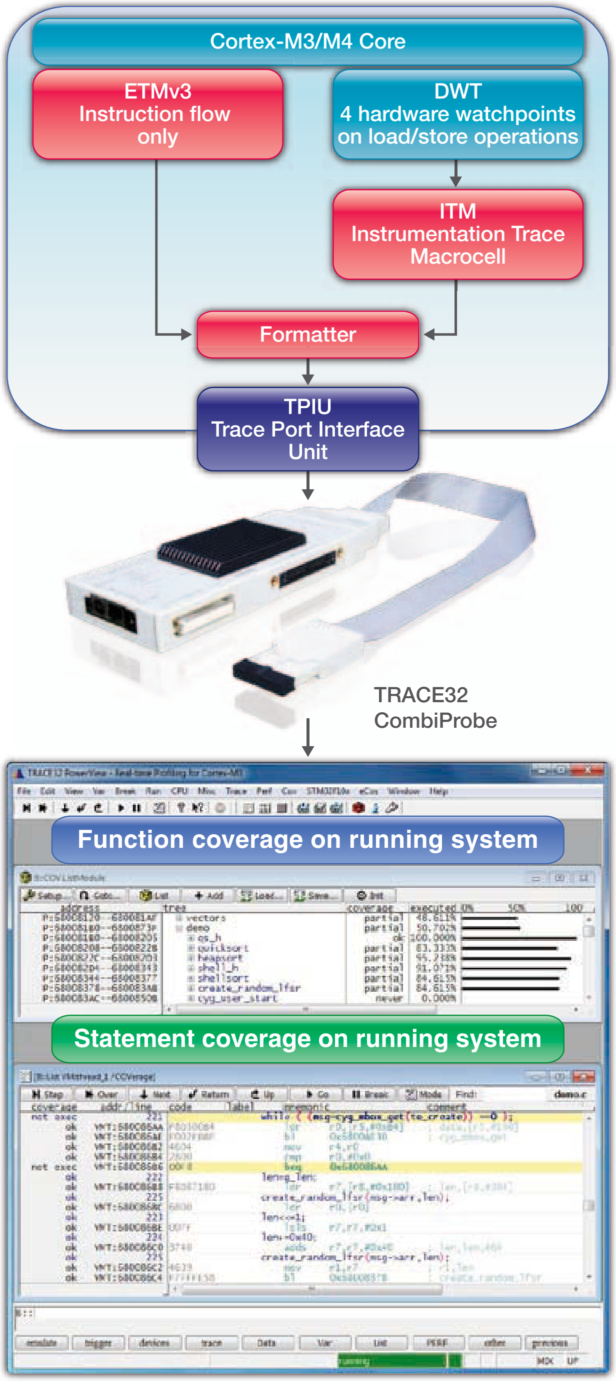

Cortex-M3/M4 Core
ETMv3 Instruction flow only
DWT 4 hardware watchpoints on load/store operations
ITM Instrumentation Trace Macrocell
Formatter
TPIU Trace Port Interface Unit
TRACE32 CombiProbe
Function coverage on running system
Statement coverage on running system

Fig. 17: Real-time profiling enables code-coverage analysis to be followed live on the screen

Real-time Profiling is particularly suitable for performing statement and condition coverage. The coverage analysis can be followed live on the screen and the test results are visible immediately (see figure17). “ok” marked lines are already covered.

Simulation and Reality Draw Closer Together

It is now common to perform simulation and verification of designs before committing to hardware. This is why tools such as MATLAB® and Simulink® have made inroads as development software into the control engineering market. It can save a lot of time and effort if the control loop can be tested for the effects of many variables before finalizing the design.

So what is the next step, after the control algorithm has been found through simulation? How is this solution integrated into the control hardware? For this, Simulink enables you to generate code automatically. But can you be sure that the program behaves the same way on the control hardware as in the simulation?

Verification Approach

The Institute of Flight System Dynamics at Technische Universität München came up with an interesting solution during development of a flight control system for a Diamond DA42 (see figure 20).

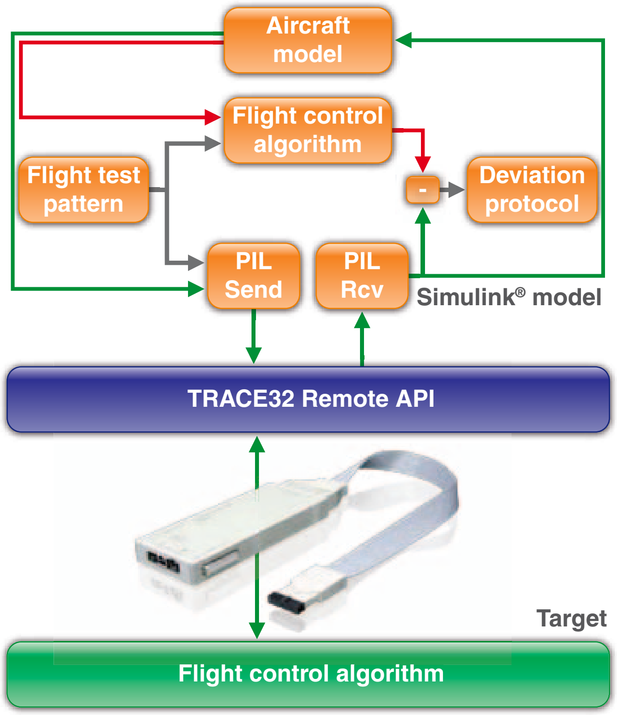


Fig. 18: The real control behavior (green path) and the simulated control behavior (red path) are compared.

After the control algorithms had been created and functionally tested with Simulink, the corresponding program code for the processor of the control hardware was generated from the control blocks using the Embedded Coder. Using a TRACE32 debugger, the generated code was loaded into the control hardware and functionally tested in-situ.

To determine the level of deviation between simulated control behavior (red path) and real control behavior (green path), but above all to confirm the numeric accuracy of the control hardware, a Processor-In-the-Loop simulation (PIL) was chosen (see Figure 18). Essentially, the PIL simulation is based on the specially developed Simulink blocks "PIL Send" and "PIL Receive". These were designed to implement communication between Simulink and the TRACE32 Remote API.

In each run through, the flight control algorithm performs a single calculation step of the discrete time flight control on the target hardware. The Simulink model provides the necessary input parameters. The values calculated are returned to the Simulink model and there supply the aircraft model. In a parallel calculation, the simulated flight control algorithm computes the same values. The difference is then used to compare the two results.

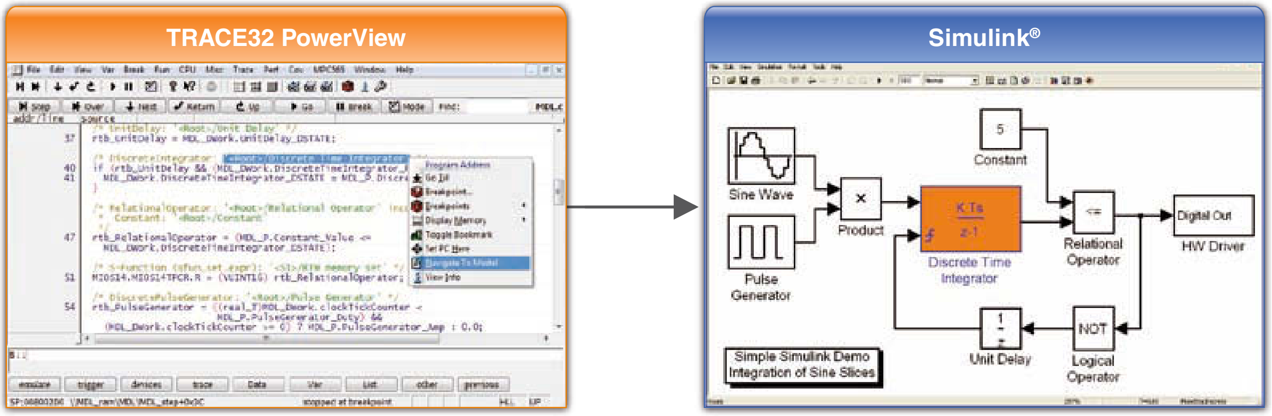
The testing in the stand resulted in an absolute deviation of 10^{-13} – a high level of consistency that was elegantly and easily proven with this approach.

For more information about the project of the Institute of Flight System Dynamics at the Technische Universität München, go to **www.lauterbach.com/intsimulink.html.**

TRACE32 Integration for Simulink®

At the Embedded World show February 2012 in Nuremberg/Germany, Lauterbach will be presenting an even closer coupling between Simulink and Lauterbach's TRACE32 debuggers.

Lauterbach has used the property of the Simulink code generation that the code block always begins with a comment line which contains the name and model path for the block. These comment lines are available after the generated code has been loaded into the



```
/* UnitDelay: '<Root>/Unit Delay' */
rtb_UnitDelay = MDL_DWork.UnitDelay_DSTATE;

/* DiscreteIntegrator: '<Root>/Discrete Time Integrator'
if (rtb_UnitDelay && (MDL_DWork.DiscreteTimeIntegrator_
  MDL_DWork.DiscreteTimeIntegrator_DSTATE = MDL_P.Discre
}

/* RelationalOperator: '<Root>/Relational Operator' inc
 *  Constant: '<Root>/Constant'
 */
rtb_RelationalOperator = (MDL_P.Constant_Value <=
  MDL_DWork.DiscreteTimeIntegrator_DSTATE);

/* S-Function (sfun_set_expr): '<S1>/RTW memory set' */
MIOS14.MIOS14TPCR.R = (VUINT16) rtb_RelationalOperator;

/* DiscretePulseGenerator: '<Root>/Pulse Generator' */
rtb_PulseGenerator = ((real_T)MDL_DWork.clockTickCounter <
                      MDL_P.PulseGenerator_Duty) &&
  (MDL_DWork.clockTickCounter >= 0) ? MDL_P.PulseGenerator_Amp : 0.0;
```

Fig. 19: The block belonging to the selected source code line is marked in Simulink.

TRACE32 debugger. These lines allow a simple correlation between the Simulink block and the lines in the source code.

Navigation from Simulink® to TRACE32

A global TRACE32 menu and TRACE32 menus for blocks and signals are integrated into Simulink as 'Simulink Customization Menus'. The TRACE32 debugger can be controlled from Simulink with the help of these menus. The following functions are available:

- Show block code in TRACE32
- Open TRACE32 Variable Watch Window for signals
- Load Simulink build to the TRACE32 debugger
- Set and manage block/signal breakpoints
- Start and stop program on the control hardware

Navigation from TRACE32 to Simulink®

Selecting a section of source code in the TRACE32 debugger marks the corresponding block in Simulink (see Figure 19).

Future

When Simulink Release 2012a is available, further TRACE32 functions will be possible in Simulink. Lauterbach will use the improved functionality of the Simulink rtiostream API to integrate a PIL simulation, data logging, and parameter tuning.

MATLAB® and Simulink® are registered trademarks of The MathWorks, Inc.

Fig. 20: Diamond DA42 (Source: www.diamond-air.at)

UEFI BIOS Debugging with TRACE32

A new TRACE32 extension for the Atom™ Debugger provides a complete debug capability of Insyde's H2O UEFI BIOS.

UEFI is the successor to the traditional PC BIOS. It functions as an interface between firmware and operating system managing the boot process. From power-on to takeover by the operating system, UEFI runs through various, clearly distinguished phases (see figure 21).

As it is a JTAG-based tool, TRACE32 allows debugging to start from the reset vector.

In each phase of the boot process, the PowerView user interface provides special windows which show UEFI specific information. Functions and prepared scripts enable debugging of dynamically loaded drivers starting from the first instruction. For more information about the new UEFI extension, go to **www.lauterbach.com/uefi.html**.

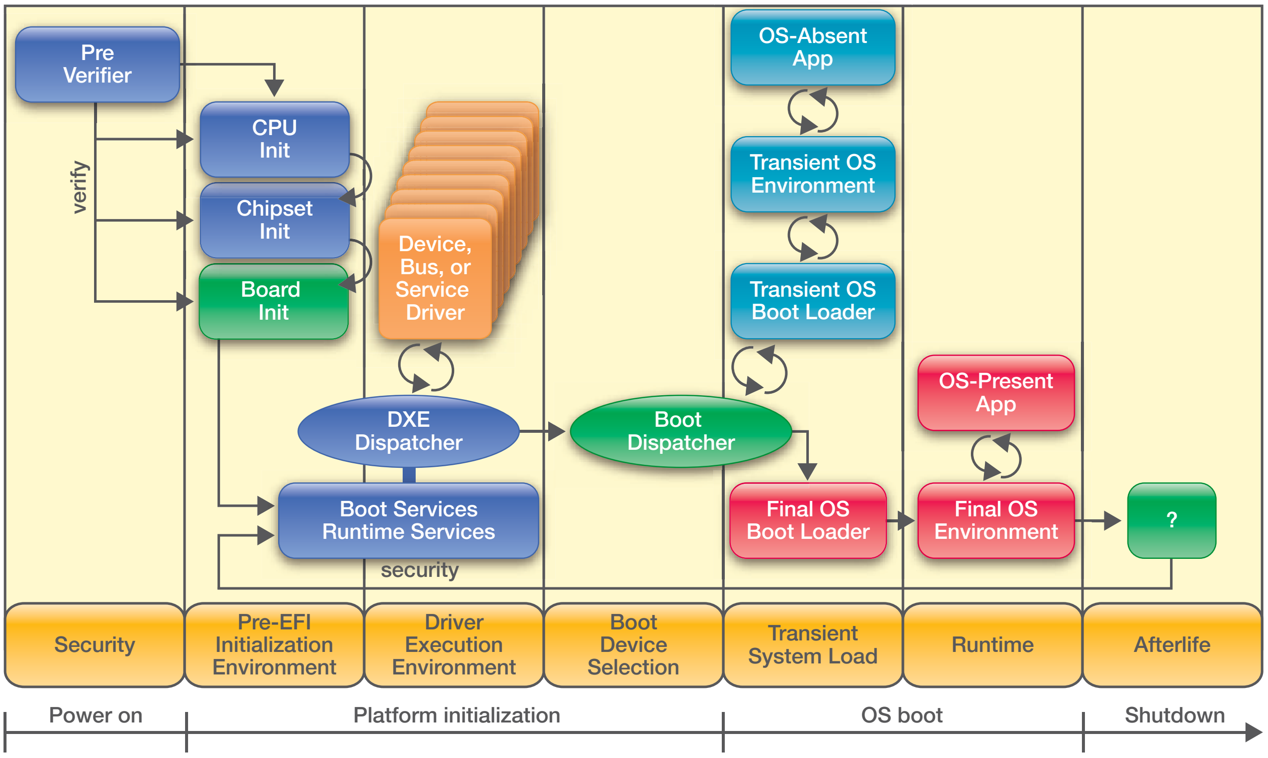


Fig. 21: System boot process with UEFI.

WORLDWIDE BRANCHES

- USA
- Germany
- France
- UK
- Italy
- China
- Japan

Represented by experienced partners in all other countries

KEEP US INFORMED

If your address has changed or if you no longer want to be on our mailing list, please send us an e-mail to:

info_us@lauterbach.com

Not in Kansas anymore: Securing SCADA

Security researchers warn that attacks against supervisory control and data acquisition (SCADA) systems could cripple critical infrastructure services. SCADA networks encompass computers and applications that perform key functions in providing essential services and commodities such as electricity, natural gas, gasoline, water, waste treatment, and transportation—all part of the nation's critical infrastructure. The first step in safeguarding our critical infrastructures is in identifying system vulnerabilities.

Even though SCADA systems have been used for a decade to monitor and control critical equipment at power companies, manufacturing facilities, water treatment plants, and even building automation, not until recently has the focus been on security and the vulnerabilities of such systems.

SYSTEM VULNERABILITIES

Digital Bond, a consulting firm specializing in control-system security, has found that the latest vulnerabilities mostly exist in free or low-cost Windows-based engineering work-stations that are used as graphical user interfaces to back-end control systems. SCADA systems such as Siemens are deployed widely in critical infrastructures.

Siemens reported last year that a Stuxnet worm was released for the purpose of stealing industrial secrets, disturbing operations and infecting some 14 nuclear plants. The worm leveraged a previously unknown Windows vulnerability (now patched) that allowed it to spread from computer to computer, typically via USB sticks. In today's times, it has become increasing apparent that attacks on vulnerable SCADA systems can wreak havoc.

! SCADA is not secure, but what can be done to establish security?

Cambashi analyst Christine Easterfield agrees and advises that you "consider operational procedures, staff, and other factors. For example, staff need to be trained in secure practices and made aware of the risks to which they may expose critical systems."

Blue Pillar, a provider of energy assets management software, confirmed Cambashi's operational procedures and staff concerns and believes that with the exception of the IT staff, the operational and energy management staff does not even have energy asset security on their radar as a security concern. The reality is that they either rely 100% on physical security or they have to rely on the unsecured and open industrial automation implementations running Modbus TCP-IP throughout their networks.

According to Kyle Zeronik, Blue Pillar's VP of Information Technology, it's critical to secure the SCADA from top to bottom. "We secure critical power infrastructures right down to securing the messaging within our architecture to limit the conversations to only the devices with appropriate credentials and authorizations. We manage site-site communication including Internet security and encrypted messages transmitted over secure channels. Device level communications is managed via 256-bit AES (FIPS-197 certified) encryption."

SECURITY CHECKS AND BALANCES

Any facility with a connection to the SCADA system should conduct a physical security survey and inventory access point check. It's imperative to identify and assess any source of information including remote telephone, computer network, and fiber optic cables that could be tapped; radio and microwave links that are exploitable; computer terminals that could be accessed; and wireless local area network access points. The goal is to identify and eliminate single points of failure.

The National Infrastructure Protection Plan Program works with several government agencies in the area of cyber security to ensure the integrity and availability of the nation's cyber infrastructure. In addition, the National Supervisory Control and Data Acquisition (SCADA) Test Bed is a DOE Office of Electricity Delivery and Energy Reliability (OE)-sponsored resource to help secure our nation's energy control systems. It combines state-of-the-art operational system testing facilities with research, development, and training to discover and address critical security vulnerabilities and threats to the energy sector. ■

Eric Marks is the industry practice leader for PricewaterhouseCoopers. He holds a bachelor of mathematics in computer science from the University of Waterloo and an MBA in strategic management and marketing from The Wharton School of the University of Pennsylvania.

This excerpt is from a longer article at www.eetimes.com/4238057.

designwest

Speaker

David Kalinsky is a teacher of intensive short courses on embedded systems and software development for professional engineers. One of his popular courses is "Introduction to Software Security for Embedded." His courses are presented regularly in open-class format at technical training providers in international locations such as Munich, Singapore, Stockholm, and Tel-Aviv, as well as in his "home market" of the USA. See *www.kalinskyassociates.com*.

Even if your device is not connected to the Internet, you need to protect it from malicious attacks. Here are some simple protections you can institute to make your system more impenetrable.

Security fundamentals for embedded software

BY DAVID KALINSKY

I was preparing for a trip to the Eastern European city where my parents had lived as children. I had never been there. I googled the name of the city, and was quickly led to a story that was surprising and chilling: A high school student there had modified a TV remote control so that it could control the city's tram system—thus converting the urban railways into his own giant model train set. While switching tracks using his infrared gadget, this kid caused trams to derail. Twelve people were injured in one derailment.[1]

Recently, new terms like *Stuxnet* and *Duqu* have entered our lexicon. Embedded systems including those that do supervisory control and data acquisition (SCADA) are under relentless security attacks.

Many embedded software developers feel that embedded systems security should be handled at the systems-engineering level or by the hardware that surrounds their software. And indeed many things can be done at those levels, including:

- Secure network communication protocols.
- Firewalls.
- Data encryption.
- Authentication of data sources.
- Hardware-assisted control-flow monitoring.

But these traditional techniques aren't enough, as was frighteningly described at last year's DesignCon East 2011 talk "Strong Encryption and Correct Design are Not Enough: Protecting Your Secure System from Side Channel

Attacks." The speaker outlined how power consumption measurements, electromagnetic leaks, acoustic emissions, and timing measurements can give attackers information they can use to attack your embedded device.

Clearly then, system-level and hardware defenses are not enough. Most security attacks are known to exploit vulnerabilities within application software. Vulnerabilities are introduced into our embedded systems during software design and development. Since system-level and hardware defenses against security attacks are far from perfect, we need to build a third line of defense by dealing with vulnerabilities in our application software.

If an attacker succeeds in breaking into software running at a high level of privilege, immediately your attacker will be operating at a high level of privilege too.

While our software line of defense will surely be less than perfect, we need to work on that line of defense with the immediate objective of reducing the size of the "attack windows" that exist in our software. The very first step in doing this is to try to think like an attacker. Ask how an attacker could exploit your system and your software in order to penetrate it. You might call this a *threat analysis*. Use the results to describe what your software should not do. You might call those *abuse cases*. Use them to plan how to make your software better resist, tolerate or recover from attacks.

Don't forget that our attackers have a big advantage when it comes to embedded systems: Most embedded software has severe execution time constraints, often a mixture of hard real-time and soft real-time tasks. This coaxes us to design application software that is "lean and mean," by reducing to a minimum intensive run-time limit checking and reasonableness checking (for example, invariant assertions) in order to meet timing requirements. Our attackers have no such execution time constraints: They are perfectly happy to spend perhaps weeks or months researching, preparing, and running their attacks—possibly trying the same attack millions of times in the hope that one of those times it might succeed, or possibly trying a different attack each day until one hits an open "attack window."

HOW CAN ATTACKERS ATTACK VIA OUR OWN SOFTWARE?

Quite often embedded software developers dismiss the issue of embedded software security, saying: "Hey, our device will never connect to the Internet or to any other external communication link. So we're immune to attack." Unfortunately, this is naïve and untrue. I'd like to present a counterexample:

Many embedded devices use analog-to-digital-converters (ADCs) for data acquisition. These ADCs may be sampled on a regular timed basis, and the data samples stored by application software in an array. Application software later processes the array of data. But an attacker could view this in a totally different way: "What if I fed the ADC with electrical signals that, when sampled, would be exactly the hexadecimal representation of executable code of a nasty program I could write?" In that way, the attacker could inject some of his software into your computer. No network or Internet needed.

Seems like a lot of work to build an "ADC Code Injector" device just for this purpose. But the attacker might not be just a high-school kid. He might be a big industrial espionage lab, or a large, well-funded team working at the national laboratory of a foreign government.

Now, how could he get your processor to execute his program that he's injected? He might gamble that your software stores the ADC data array on a stack (perhaps using `alloca()` or `malloc()`). If his luck is good, he could cause an array overflow, possibly by toying with the hardware timer that controls the ADC data sampling. A typical normal stack layout is shown in **Figure 1**.

If the attacker succeeds in causing an array overflow, the stack could become corrupted, as shown in **Figure 2** below. Note that "return address" was stored on the stack at a location beyond the end of the array.

If the attacker plans the corruption just right, the overflow will reach the location on the stack where the current re-

Typical normal stack layout.

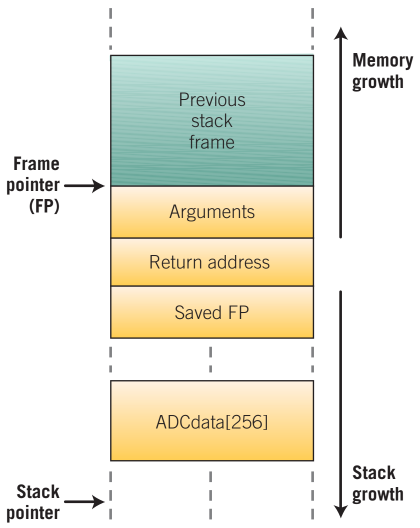


Figure 1

Stack is corrupted after array overflow.

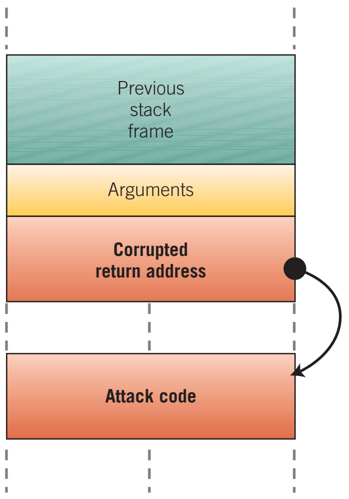


Figure 2

turn address was stored. This can be used to insert into this stack location a pointer to his own code. As a result, when "Return Address" is used by your code, control will pass to the attacker's code. Suddenly *his* code is executing on your processor, instead of your code.

This is called a *stack smashing* attack. Please note that it was done in this example without an Internet connection, and without a connection to any external communication line.

Of course, it could have been helpful for our attacker to have the source code for your embedded software—as a disgruntled ex-employee might. But I think a patient and resourceful attacker team could develop this kind of attack even without your source code.

Can you think of an easier way for an attacker to develop an attack on your current project ?

WHAT'S SW DEVELOPER TO DO?
During embedded systems software design, you can enhance software security by keeping several fundamental ideas in mind:[2]

**Mindframe #1:
Distrustful decomposition**
Separate the functionality of your software into *mutually untrusting chunks*, so as to shrink the attack windows into each chunk. (In embedded software, we sometimes call these chunks *processes* or *subsystems* or *CSCIs*.)

Design each chunk under the assumption that other software chunks with which it interacts have been attacked, and it is attacker software rather than normal application software that is running in those interacting chunks. Do not trust the results of interacting chunks. Do not expose your data to other chunks via shared memory. Use orderly inter-process communication mechanisms instead, like operating system message queues, sockets, or TIPC (Transparent Inter-process Communication). Check the content you receive.

As a result of mutually untrusting chunking, your entire system will not be given into the hands of an attacker if any one of its chunks has been compromised.

Mindframe #2: Privilege separation
Keep to a minimum the part of your code that executes with special privilege.

Think about it for a moment: If an attacker succeeds in breaking into software that's running at a high level of privilege, immediately your attacker will be operating at a high level of privilege too. That'll give him an extra-wide open "attack window" into your system.

So let's avoid running application software in kernel mode, or master mode, or supervisor mode, or whatever your particular CPU architecture may

! Some of us would say these are bugs. But I'd like to call ! them vulnerabilities here . . . Small vulnerabilities can ! open the window to huge attacks.

call it. Leave that mode for operating system use only. Run your application software strictly in user mode. This will enlist your CPU hardware in efforts to limit your software's attack window.

**Mindframe #3:
Clear sensitive information**
Clear every reusable resource when freeing it.

Think about it: After you've released the resource, be it a RAM buffer or a software-hardware interface data register, the next user of the very same resource might be an attacker. Embedded system attackers enjoy "phishing" these resources, just as much as Internet attackers enjoy phishing. Wouldn't they be overjoyed to read whatever data you had been working on in the buffer, or to read the data you've just given to hardware for output!

SEGGER

Emulator (J-Link™)
ARM and Cortex

- USB to JTAG Emulator
- Fast 720kb/s Download Speed
- Serial Wire Debug (SWD) Support
- Multicore Debugging Support
- Auto JTAG Speed Recognition

8.12.00 J-Link ARM PRO
Adds Ethernet Conectivity and Licensing for All Enhancement Modules

The J-Link can be coupled with a number of available software modules to fit your application needs.

J-Flash is a stand-alone application used with the J-Link to program internal and external flash devices.

J-Link RDI permits the use of the J-Link with an RDI compliant debugger.

J-Link GDB Server is a remote server for GDB.

J-Link Flash Breakpoint permits you to set an unlimited number of software breakpoints while debugging in flash.

As the original manufacturer of the J-Link and its OEM derivatives, we are happy to inform you that this software also supports the DIGI® JTAG Link, Atmel® SAM-ICE, and IAR® J-Link KS.

$60
Special Offer

8.08.90 J-Link EDU
Educational Use J-Link
Includes Flash Breakpoints
Includes J-Link GDB Server
www.segger-us.com/edu.html

J-Link Educational Use (EDU) Bundle

www.segger.com

Most resource release services in embedded environments simply mark the newly freed resource as "available." They leave the old information contained in the resource potentially visible to new users, trusting the new user to over-write the old content rather than reading it. This is done since it's much faster than explicitly nulling out the resource.

So when an application is done with a resource, it's up to the application to prepare for releasing the resource by first zeroing out each and every:

- Heap buffer, memory pool buffer, memory partition segment.
- Statically-allocated memory buffer.
- Released stack area.
- Memory cache.
- File in a file system.
- Hardware interface data register, status register, control register.

designwest
Speaker

ESC 2012 CLASSES DAVID KALINSKY

Dave Kalinsky will teach four classes at ESC/DESIGN West in March, 2012. Also look for classes he will be teaching at ESC/DESIGN East in Boston this year.

ESC-101: Software Design for Multicore Systems—2012 EDITION
8:00 AM to 12:00 PM,
March 26

ESC-402: Is Static Code Analysis Ready for Real-time?
3:15 PM–4:15 PM,
March 29

WHAT CAN BE DONE DURING CODING?

During embedded systems programming, developers can augment software security by avoiding a number of common software security vulnerabilities.

Some of us would say these are bugs. But I'd like to call them *vulnerabilities* here, to emphasize that some tiny software "defect" that might be too minor even to be called a bug—might be just what an attacker is looking for in order to mount his attack on your embedded system. Small vulnerabilities can open the window to huge attacks.

Vulnerability #1: Buffer overflow

Far and away, the most widespread security vulnerability in C-language coding is buffer overflow. It could be as simple as writing into element number 256 of a 256-element array.

Compilers don't always identify out-of-bounds buffer access as a software defect. Yet buffer overflow can lead to more serious consequences, such as stack smashing that was discussed earlier, code injection, or even *arc injection*—by which an attacker changes the control flow of your program by modifying the return address on stack. In arc injection, an attacker doesn't even have to inject any code, and he can jump to an arbitrary function in existing code, or bypass validity checks or assertions.

Here's an example of a buffer overflow attack: An embedded device is required to measure the temperature of water in a swimming pool and to display a histogram showing the percentage of time that the water is at various temperatures. The software developer creates an array of 100 positive integers, each element corresponding to one degree Celsius. Element 0 for 0°C. Element 1 for 1°C, etc. Each time the temperature sensor makes a water temperature measurement, the corresponding element of the array is incremented by 1.

Remember, this is a swimming pool to be used by humans. So the programmer feels safe and secure in designing his temperature array with lots and lots of room beyond the range of water temperature values that a human body can tolerate.

Until one day, an attacker pulls the temperature sensor out of the water and heats it up using a cigarette lighter. As soon as the sensor measures a value greater than 100°C, the histogram update software corrupts an address in memory beyond the end of the temperature array. If there's data there, the attacker will have corrupted the data. If there's machine code there, the attacker will have corrupted the executable software. In either case, this is a damaging attack. Please note (once again) that it was done without an Internet connection, and without a connection to any external communication line. Just a cigarette lighter.

Please note (once again) that it was done without an Internet connection, and without a connection to any external communication line. Just a cigarette lighter.

How can we avoid buffer overflows? This vulnerability is so widespread (and so widely sought-after by attackers), that a multipronged approach is best: prevent, detect, and recover. Prevent buffer overflows by careful input validation: check that a temperature sensor is reporting a value within bounds. In our swimming pool example, explicitly check that it's not reporting a temperature that corresponds to ice or to superheated vapor (> 100°C).

Prevent buffer overflows also by avoiding dangerous library functions (like `gets()`) and exercising extra care with others (like `memcpy()`).

Detect buffer overflows by using the idea of "paint": Extend the buffer slightly at both ends. Fill the extension areas with unusual content I call "paint"; for example, a trap instruction in your processor's machine language. Then check the paint repeatedly at run time. If the paint has been over-written, you've detected a buffer overflow.

Vulnerability #2: Pointer shenanigans
If an attacker can modify a data pointer, then the attacker can point to wherever he likes and write whatever he likes. If an attacker can over-write a function pointer, the attacker is well on his way to executing his code on your processor.

Vulnerability #3: Dynamic memory allocation flaws
It's so easy to write defective code for dynamic memory allocation, that the use of dynamic memory allocation is forbidden in many embedded aerospace and safety-critical systems. Of course, attackers are eager to search out these defects, as they also represent golden opportunities for them to violate the security of an embedded system.

Common flaws include double-freeing, referencing of freed memory, writing to freed memory, zero-length allocations, and buffer overflows (again).

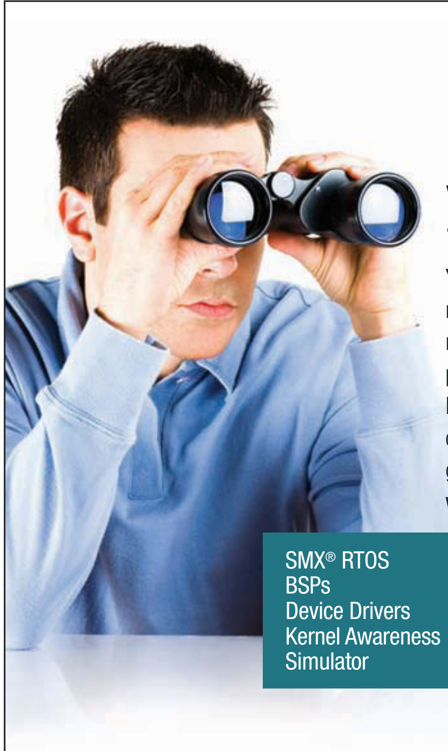
A flaw that is particularly sensitive in embedded software, is neglecting to check the success or failure of a memory allocation request. Some memory allocators will return a zero instead of a pointer to a memory buffer, if they run out of available memory. If application software treats this zero as a pointer, it will then begin writing to what it thinks is a buffer starting at memory address zero.

Many an attacker would be happy to have your software do this. Attackers know that embedded systems tend to be tightly memory constrained. They will try to make a system run out of memory by doing whatever they can to force your memory allocator to allocate more memory than usual—perhaps by leaking memory, possibly leaking it into some code they've injected. They may also try to flood your data-acquisition system with higher than normal volumes of data or higher rates of data—in the hope that the avalanche of data will exhaust your memory capacity. And then … if your software asks for a buffer but neglects to check for allocation failure, it will begin writing a buffer at address zero—trampling upon whatever was there. For example, if your interrupt enable/disable flags happen to be at that address, this could turn off the connection between software and peripheral hardware interfaces. Essentially, this could dis-embed your embedded system.

! ! ! Data entering an embedded system from the outside world must not be trusted. It must be "sanitized" before use. This is true for even the simplest of integers.

Vulnerability #4: Tainted data
Data entering an embedded system from the outside world must not be trusted. Instead, it must be "sanitized" before use.

This is true for all kinds of data streams as well as even the simplest of integers. Attackers are on the lookout for extreme values that will produce abnormal effects. In particular they're looking for unexpected values, like situations where a digital microprocessor would give a different result from what a human would calculate using pencil and paper. For example, an integer 'i' happens to have the value 2,147,483,647. If I were to add 1 to this value in a back-of-the-envelope calculation, I'd get +2,147,483,648. But if my microproces-



Looking for the right software for your processor?

You've found it. Our software is built to run right out of the box—no integration required—and we provide full support for popular tools. With Micro Digital you have low-cost, no-royalty licensing, full source code, and direct programmer support. So get your project off to the right start. Visit **www.smxrtos.com/processors** today.

SMX® RTOS	TCP/IP	USB Device
BSPs	FAT File System	USB Host
Device Drivers	Flash File System	USB OTG
Kernel Awareness	GUI	WiFi
Simulator	C++ Support	Floating Point

Free Evaluation Kits: www.smxrtos.com/eval
Free Demos: www.smxrtos.com/demo

Micro Digital
RTOS INNOVATORS
www.smxrtos.com

ARM • Cortex • ColdFire • PowerPC • x86 • IAR EWARM • CrossWorks • GCC • CodeWarrior

sor were to execute i++, it would get -2,147,483,648 (a large negative number). It wouldn't take long for a clever attacker to leverage this kind of quirk into some kind of havoc in an embedded system.

A useful technique for data sanitization is called *white listing*. It involves

! A determined attacker can undermine an embedded system—even one without an Internet connection, and without a connection to any external communication line.

describing all possible valid values for a given piece of data and then writing code that only accepts those values. All unexpected values are viewed as "tainted" and are not used.

MORE TO EXPLORE

Clearly the concepts of software security for embedded systems are not limited to three design "mindframes" and four coding "vulnerabilities." Attackers are a creative bunch, always finding new ways to threaten the security of our software and systems. The story of software security is incessantly changing. One way to keep up is to visit a website called CWE—Common Weakness Enumeration (*http://cwe.mitre.org/*) that keeps a continually updated list of software weaknesses for security.[3] Most of them are as relevant for embedded software as for non-embedded software. I'd start at their "Top 25 Most Dangerous Software Errors" list, which is updated each year.

We've seen several ways that a determined attacker can undermine an embedded system—even one without an Internet connection, and without a connection to any external communication line. We've also seen that embedded software designers and programmers can contribute an additional layer of defense against malicious attacks, beyond what can be done at system-level and in hardware. The embedded software community needs to be alert to the special "mindframes" and "vulnerabilities" involved in this new challenge to our embedded systems. ■

David Kalinsky is director of customer education at D. Kalinsky Associates—Technical Training, a provider of intensive short courses on embedded systems and software development for professional engineers. He is a popular lecturer and seminar leader on technologies for embedded software in North America, Europe, and Israel. In recent years, David has built high-tech training programs for a number of Silicon Valley companies, on various real-time operating systems and other aspects of software engineering for the development of real-time and embedded systems. Before that, he was involved in the design of many embedded medical and aerospace systems. David holds a Ph.D. in nuclear physics from Yale University. Contact him through *www.kalinskyassociates.com*.

ENDNOTES

1. "Daily Telegraph" newspaper London UK, "Schoolboy hacks into city's tram system." The city: Lodz, Poland. The date: 11 Jan. 2008.
2. Dougherty, C., K. Sayre, R.C. Seacord, D. Svoboda, and K. Togashi. "Secure Design Patterns," CERT Program, Software Engineering Institute, Carnegie Mellon University, Pittsburgh PA, Technical Report CMU/SEI-2009-TR-101, ESC-TR-2009-010.
3. CWE's "Common Weakness Enumeration," The MITRE Corporation, Bedford MA, *cwe.mitre.org*.

A lifetime of support

TRACE32®

www.lauterbach.com

LAUTERBACH DEVELOPMENT TOOLS

Embedded systems designers can protect sensitive data that's on a device's hard drive (data-at-rest) by using encryption techniques.

Enhance system security with better data-at-rest encryption

BY DAVID KLEIDERMACHER, GREEN HILLS SOFTWARE

In 2010, the television network CBS aired a program demonstrating how discarded office copiers are gold mines for private information, trivially harvested from disk drives within the machines.[1] From copiers randomly selected from a used copier warehouse, investigators recovered lists of wanted sex offenders, drug-raid targets, architectural design plans, personal identification information (name, address, Social Security number), and medical records—including blood-test results and a cancer diagnosis.

When asked whether this could be prevented, one copier company said that customers could purchase a $500 option that will erase copied images from the hard drive after use. Give the guy who wrote those couple lines of code a bonus!

Another obvious solution to this problem is data-at-rest protection. *Data-at-rest protection* is a when data stored on a device and not in transit, known as *data at rest*, is either encrypted or follows certain protocols that include encryption to protect the data from unauthorized access. The storage media for an embedded system may include hard disk drives, flash memory, and attached USB thumb drives. As witnessed by the photo copier story, seemingly benign, mundane office equipment is often vulnerable and not protected. On the other hand, many modern embedded systems do have encrypted storage-protection requirements, driven by intellectual property protection, digital rights management, sensitive customer information, and more. Compliance regulations in certain industries require that sensitive stored data be protected with appropriate data-protection protocols that include encryption. See sidebar for examples.

This article discusses approaches for protecting data-at-rest.

CHOOSING THE STORAGE LAYER

As shown in **Figure 1**, developers may choose from multiple layers in the data-storage stack to apply data-at-rest protection protocols.

Hardware layer: With *full-disk encryption (FDE)*, the entire medium used for storage is encrypted. All the data that goes on the storage medium is encrypted, including certain hidden files, such as the operating system's temporary files and swap space. The advantage is such files are not exposed. However, the drive itself is not encrypted, leaving the master boot record exposed.

When FDE is handled within the medium peripheral itself, it's referred to as a *self-encrypting drive (SED)*. SEDs are common in the laptop market. The advantage of SEDs for the embedded systems developer is that little or no new software must be written to take advantage of the data-protection facilities. Encryption is performed with specialized hardware within the storage device, offloading the main embedded applications processor. If self-encrypting storage media is feasible, it's an excellent choice due to ease of use, excellent performance, and the ability to hide the storage encryption key from the main applications processor and memory. Unfortunately, many embedded systems will be unable to use the available standalone SED products due to form-factor limitations.

Block manager layer: Encryption can be performed at the next level up, the device-management layer, typically a block-oriented driver. Protection at this level may cover the entire managed device (FDE). The performance implications of this approach vary. If the embedded platform contains a symmetric encryption accelerator, the overhead is likely to be reasonable, while a purely software cryptographic implementation may cause a dramatic loss in performance. Embedded systems developers can architect the encryption facilities such that the device driver calls out to generic medium block encryption routines, ensuring that software is easier to maintain across different generations of the embedded product that may use different types of storage.

File system layer: The next candidate for data-at-rest protection is the file system. The major advantage of implementing storage protection at the file system layer is to provide finer granularity over the choice of information that requires storage confidentiality. This is especially important if encryption is performed in software with minimal or no hardware acceleration. Depending on the file system implementation, developers may be provided options for encryption at the volume level or at the individual file level.

Applications layer: Finally, applications can add their own data protection, either using underlying file-system encryption features or a custom implementation. For example, an audit logging application can encrypt its audit records prior to calling the standard file system output functions.

For volume, file, or application-level data protection, developers can employ separate keys for these groups of data rather than a single key for the entire system. This is a sensible application of "least privilege" principles.

Developers resorting to custom, application-level approaches will also need to design their own key-management system, whereas users of encrypting file systems or SEDs can use the key-management framework provided by the product supplier.

COMPLIANCE REGULATIONS

- **Medical sector:** the Health Industry Portability and Accounting Act (HIPAA) requires that patient data stored within medical devices is protected.
- **Financial sector:** the Payment Card Industry (PCI) data security standard (PCI DSS) requires the protection of credit card information within financial processing systems.
- **Government and security-conscious enterprises:** Data-at-rest protection within smartphones and tablets is a requirement if handhelds are used for the processing of sensitive information.

WHICH ENCRYPTION ALGORITHM?

Data-at-rest presents some unique challenges for encryption algorithms relative to network security protocols.

For data-at-rest protection, an encryption algorithm must be performed without adding additional storage space: A plaintext media block is encrypted in place, generating a ciphertext block of the same size. The most basic encryption mode, **electronic code book (ECB)**, would provide this memory conservation but is not suitable for data-at-rest encryption since any two same plaintext blocks will encrypt to the same cipher-

Data-at-rest protection choices by layer.

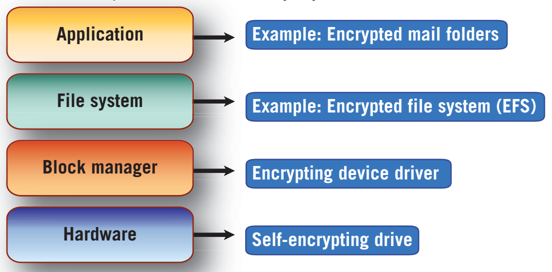


Figure 1

Tweakable block cipher overview.

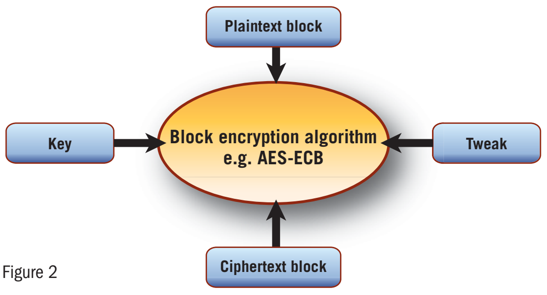


Figure 2

text, making it easy for an attacker to find patterns in the data and potentially derive information. We must consider other modes, most of which require an *initialization vector (IV)*. However, to avoid space expansion, the data-protection system must include a means for implicitly deriving this IV.

Implicit IV derivation poses a surprisingly difficult challenge for common encryption modes. Many modes require uniqueness: The same IV must never be reused for a particular key. For example, with **counter mode**, a predictable counter can be used, but the same number can never be repeated for a given key. For **cipher block chaining (CBC) mode**, a unique and unpredictable number must be used. Network security protocols have the freedom to generate the IV and send it along as part of the transmitted data; for the **Advanced Encryption Standard with CBC (AES-CBC)**, each transmission can generate a new random number for the IV and transmit this IV to the receiver. But for data-at-rest, we have no room to store the IV for subsequent decryption.

The obvious source for an implicit IV would be the sector number and offset for a particular data block. Using this combination provides every disk block with a unique input value. However, as data is read and written over time, the same sector and offset are reused for the same key. This implies a serious weakness in the applicability of common encryption modes for data-at-rest protection. Numerous other weaknesses of common modes, especially CBC, have been identified when applied to data-at-rest protection protocols. Clemens Fruhwirth has written an excellent paper discussing these weaknesses.[2]

TWEAKABLE CIPHERS

The good news is that cryptographers have worked diligently to address this encryption mode challenge. Liskov, Rivest, and Wagner introduced the concept of a tweakable block cipher in 2002.[3] The basic idea of a *tweakable cipher* is to apply the IV concept to the single-block cipher itself rather than to a chaining mode built on top of the block cipher. As shown **Figure 2**, the block cipher converts a plaintext block to a ciphertext block, using both the traditional key as well as the tweak as inputs.

The practical application of tweakable ciphers for the data-at-rest protection problem is the property that the cipher's security doesn't preclude reuse of the IV; thus, media sector number and block offset within the sector provide a perfect fit for tweak selection.

XTS-AES

In 2007, IEEE's Security in Storage Working Group (SISWG) published standard P1619.[4] The IEEE P1619 standard defines the **XTS-AES cipher mode** as a result of a thorough study of numerous potential tweak-based algorithms for use in data-at-rest protection.

This choice is further bolstered by NIST in "Special Publication 800-38E", which approves the XTS-AES cipher mode and references its definition in IEEE P1619-2007.[5] NIST has also amended FIPS 140-2 to include XTS-AES as an approved cipher for validation.[6]

The tweak algorithm found in XTS-AES is based on and almost identical to the one originally created by noted cryptographer Phillip Rogaway, called XEX.[7] In addition to strong security, XEX (and hence XTS-AES) are also designed for efficiency when applied to storage of many sequential data blocks (as is common with file storage).

The XTS-AES block cipher is depicted in **Figure 3.** Oddly this cipher requires twice the keying material; for 128-bit security, 256 bits of key must be used. The first half of the key is used to process the plaintext; the second half is used to encrypt a 128-bit representation of the sector number, which acts as the primary tweak, as shown in Figure 3. The result of this encryption is fed to a function that performs a Galois field multiplication (implemented as a sequence of shifts and XORs) of the encryption result with a Galois constant derived from the secondary tweak, the numeric index of the data block within the sector.

The result of this Galois multiplication is used twice. First it's added (XOR) to the plaintext block, which is then encrypted with the first key half. The Galois result is added (XOR) again to the plaintext block encryption result to create the final ciphertext block.

Decryption is similar; however, while the AES-ECB decryption algorithm is used to process the ciphertext, the tweak cipher remains the same, using the AES-ECB encryption algorithm.

In practice, data is stored to media in sectors. Therefore, the block encryption algorithm shown earlier must be executed in a loop across the entire sector. Note that while XTS-AES handles partial blocks, that part of the algorithm is often unnecessary. For example, the common sector size of 512 bytes will result in 32 block encryptions, and most media-management layers will access a full sector at a time. For such a system, given a function, `xts_encrypt`, which takes the sector number and size in bytes, plaintext block, and encryption key as input, the simple code sequence in **Listing 1** handles the sector encryption.

It's also easy to see from this code sequence that XTS-AES is parallelizable. If the embedded system contains an AES hardware accelerator (especially one that has direct support for XTS mode), this implementation should be modified to take advantage of the accelerator's ability to process multiple AES blocks at once. Furthermore, if the media allows for sector size configurability, developers may want to vary the sector size to see if bet-

The XTS-AES data-at-rest encryption cipher.

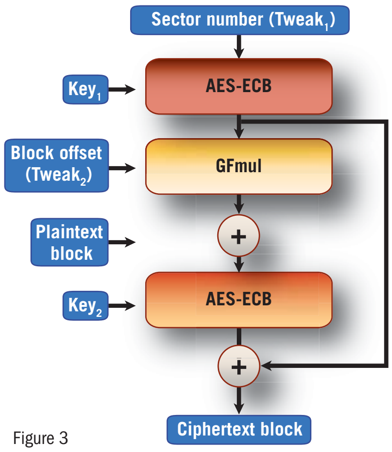


Figure 3

Listing 1

```
sector_encrypt(uint8_t *sector, uint32_t sector_num, uint32_t
        sector_size, uint8_t key[])
{
    uint32_t i;
    assert((sector_size % AES_BLOCK_SIZE) == 0);  /* 512 % 16 */
    for (i = 0; i < sector_size/AES_BLOCK_SIZE; i++)  /* 32x */
        xts_encrypt(sector+i*AES_BLOCK_SIZE, key, sector_num, i);
}
```

ter throughput (potentially at the expense of slightly reduced space efficiency) can be achieved.

When selecting data-at-rest protection products, avoid legacy approaches that use weaker modes (numerous CBC-based implementations have been commercialized). Employ the NIST- and FIPS-approved standards instead.

MANAGING THE KEY

The primary purpose of data-at-rest protection is to ensure that information residing on lost or stolen media cannot be accessed by unauthorized parties who must be assumed to have complete physical access to the disk. Thus, the symmetric storage encryption key must never be stored in the clear on the disk. However, it's often necessary to store an encrypted copy of the symmetric key on the disk (or perhaps an attached Trusted Platform Module, if available). The key is unwrapped for active use while the system is executing in an authorized manner. For personal computers such as laptops and smartphones, unwrapping is triggered by successful authentication of the user (such as using a password, smartcard, biometric, or multiple factors).

GENERATING THE KEY

A typical method of storage encryption key establishment is to convert user credentials into a key using a *key derivation function (KDF)*. A popular KDF used to convert passwords is the password-based key derivation function, version 2 (PBKDF2). PBKDF2 is defined in the RSA Laboratories' specification PKCS #5 and duplicated in RFC 2898.[8,9] PBKDF2 applies a hash function to the password concatenated with a salt (random bitstring). To make password cracking more difficult, the standard recommends that the hash output be rehashed multiple times. The recommended minimum hash iteration count is 1,000, although the number is expected to increase over time. Apple's iOS 4.0 uses 10,000 iterations. In 2010, RIM BlackBerry's encrypted backup service was determined to be vulnerable due to faulty application of PBKDF2. Instead of following the standard, the BlackBerry software used an iteration count of one.[10]

When the password is used to directly generate the storage encryption key, a change in password changes the encryption key, thereby forcing re-encryption of the entire protected media. To avoid this problem, a permanent, unique encryption key is created when the media is initially provisioned, and the key is wrapped (encrypted) with the password-derived key. With this two-level keying scheme, a periodic password change only requires rewrapping of the encryption key.

The user-authentication approach may be sufficient for limited types of attended embedded systems that can tolerate user intervention whenever the protected volumes must be unlocked. Nevertheless, this approach is not sufficient for large classes of unattended embedded systems. If the embedded system encounters a fault and automatically reboots, the encrypted volumes must be able to get back online without manual credential input.

REMOTE KEY PROVISIONING

We can consider two classes of unattended embedded systems: those that have a remote management network interface and those that do not. For the latter, the embedded system lacks any mechanism for dynamic interaction that can unlock an encryption key. In this case, if information value demands data-at-rest protection, the designer is advised to incorporate a cryptographic coprocessor that provides physical tamper-resistant key storage and internal execution of the data encryption algorithm. The device driver sends plaintext to this encryptor and receives ciphertext for storage on disk and similarly requests decryption of disk blocks as needed.

For network-enabled embedded systems, a remote management server holds a database of the provisioned data-encryption keys. A server connection is initiated by the embedded system whenever a data-encryption key must be unlocked (such as at boot time). The embedded system and server mutually authenticate, and the server provides a copy of the embedded system's provisioned data-encryption key over the secured channel.

KEY ESCROW

When implementing a data-at-rest protection system, developers must consider key escrow to guard against the possibility that the authentication information used to unlock the storage encryption key will be lost.

There are situations where the system owner may need to extract the data from storage, such as after a system failure. In most system designs, holding a copy of the data encryption key in an off-site secure location is advisable in order to prevent loss of data when the data encryption key is no longer accessible. If the embedded system lacks a network management interface, the internally-stored key must be exportable onto media for off-site escrow storage (such as in a secure vault). If the system supports network management and remote key provisioning, developers need to ensure that remotely provisioned keys are retained on a secure server or copied to protected offline media.

ADVANCED THREATS

The authentication software that runs to unlock the encrypted media must itself be trustworthy and tamper-protected. For example, the embedded operating system may incorporate the authentication function directly. The embedded operating system image (and any preceding boot loaders) is not encrypted; only the rest of the medium, which contains sensitive files, is protected. If the embedded operating system is not trusted (such as at risk of containing malware or vulnerabilities that would permit the loading of malware), the authentication process could be subverted. For example, a key logger could record the user's password, enabling recovery of the storage encryption key and all of the encrypted data.

If we assume the embedded operating system is trustworthy, we still must ensure that anything executing prior to launch of the operating system is trusted. This is a good example of the need for secure boot.

In some cases, the designer may want the embedded operating system image to be encrypted. When FDE is in use and a sophisticated operating system (such as Linux) resides on the encrypted disk, *pre-boot authentication* may be employed: a small portion of the encrypted disk contains a mini-operating system that is booted for the sole purpose of performing the authentication and unlocking the medium prior to booting the full operating system. If the embedded operating system is a secure microkernel, a separate pre-boot authentication module is not required.

Attacks against pre-boot authenticators have been successfully perpetrated. For example, the system is booted to a malicious operating system (such as a alternative booting from an external USB drive) that tampers with the pre-boot code to steal the authentication credentials as they are input.[11] Secure boot can prevent this attack as well; the signature of the modified authenticator will fail to match the known good version, aborting the boot process.

Another example of advanced threat is the *cold-boot attack*. Unless the embedded system is using a self-encrypting hard drive where the keys are stored within the media and never exposed to the main processor, disk encryption requires that the storage encryption key be kept in memory (in the clear) while the system is operational, invoking the encryption and decryption algorithm to access data. When the system is turned off, RAM is unavailable, and the only copy of the encryption key is itself encrypted. Or is it? In some systems, RAM is not immediately cleared. An attacker boots the system using a malicious operating system that grabs the plaintext key in RAM. This attack has been performed successfully.[12]

Data-at-rest protection within an embedded system equipped with secure boot and a trusted operating system impervious to remote attack can still be defeated by removing the protected media and booting it on a different computer that lacks this secure environment. Binding the storage encryption key to its intended embedded system platform can prevent this attack. In this case, the permanent storage encryption key is derived (in whole or in combination with user credentials) from a platform-specific key, such as a fused one-time programmable key or TPM key (if applicable). Even if the user's credentials are stolen, the storage encryption key cannot be derived outside of the targeted embedded platform. The downside of this extra level of defense is that a hardware failure that prevents access to the platform credential will render the data permanently inaccessible (unless the derived storage encryption key itself is securely escrowed).

! We still must ensure that anything executing prior to launch of the operating system is trusted.

PROTECT YOUR CUSTOMERS

Embedded systems developers looking to incorporate data-at-rest protection into their next designs are faced with a plethora of design choices and constraints. This article provides designers with an overview of the key issues to consider. Special considerations for data-at-rest protection include the use of government-approved symmetric encryption algorithms designed specifically for such applications and proper management of the long-term keys typically used for this purpose. ■

Dave Kleidermacher is CTO of Green Hills Software. He writes a column on Embedded.com about security issues and he teaches at the Embedded Systems Conference.

ENDNOTES

1. Keteyian, Armen. "Digital Photocopiers Loaded With Secrets" CBSnews.com, dated April 20, 2010 9:35 PM, *www.cbsnews.com/2100-18563_162-6412439.html*
2. Fruhwirth, Clements. "New Methods in Hard Disk Encryption." Institute for Computer Languages Theory and Logic Group, Vienna University of Technology, July 18, 2005.
3. Liskov, M., R. Rivest, and D. Wagner. "Tweakable Block Ciphers," 2002. MIT and UC Berkeley. *www.cs.berkeley.edu/~daw/papers/tweak-crypto02.pdf*
4. Security in Storage Working Group of the IEEE Computer Society Committee. *IEEE P1619, Standard for Cryptographic Protection of Data On Block-Oriented Storage Devices*, 2007.
5. National Institute of Standards and Technology (NIST). "NIST Special Publication 800-38E, Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices." January 2010. *csrc.nist.gov/publications/nistpubs/800-38E/nist-sp-800-38E.pdf*
6. Information Technology Laboratory, NIST. "FIPS Pub 140-2: Security Requirements For Cryptographic Modules." *csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf*
7. Rogaway, Phillip. "Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC," September 24, 2004. *www.cs.ucdavis.edu/~rogaway/papers/offsets.pdf*
8. RSA Laboratories. PKCS #5 v2.0: Password-Based Cryptography Standard. March 25, 1999.
9. PKCS #5: Password-Based Cryptography Specification Version 2.0; Internet Engineering Task Force, Request for Comments: 2898; September 2000.
10. NIST National Vulnerability Database, CVE-2010-3741. *http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2010-3741*
11. Turpe, Sven, et al. "Attacking the BitLocker Boot Process," *Proceedings of the 2nd International Conference on Trusted Computing (TRUST 2009),* Oxford, UK, April 6-8; LNCS 5471, Springer, 2009.
12. Halderman, J. Alex, et al. "Lest We Remember: Cold Boot Attacks on Encryption Keys," *Proceedings of USENIX Security '08*, pp. 45-60.

Here's a less time-consuming way to maintain magnetic card reader and card reliability in a variety of noisy electronic environments.

Make magnetic card readers more reliable in noisy environments

BY IRFAN CHAUDHRY, MAXIM INTEGRATED PRODUCTS

Plastic magnetic swipe cards are the principal means of establishing personal identity, processing financial transactions, and proving security level for access to secure corporate and military installations. Given the ubiquity of magnetic card readers (MCRs) and the harsh environments in which they're used, embedded systems designers face daunting challenges in maintaining the reliability of their MCR-based systems, especially in the face of various levels and types of electronic noise.

ONE SIZE DOES NOT FIT ALL

One of the most crucial parts of any MCR system is the magnetic read head (MRH). When a card is swiped, the MRH converts the stored data in the card's magnetic stripe to a voltage. Other MCR blocks process the converted voltage to extract the stored data. Large differences in magnetic field strengths from one card to another and differences in swipe speeds from one person to another make designing an MCR a challenging task.

Adding to the MCR design difficulty are the MRH data sheets, which do not fully specify the frequency-dependent components and are often vague when specifying other key parameters. In some cases the data-sheet specifications of two similar heads from two different manufacturers differ significantly in the list of parameters specified and those omitted. These differences are especially troublesome when you are trying to minimize noise issues in an MCR system and make designing an optimum card-reading system difficult and time-consuming.

This article outlines a strategy to resolve these specification issues, and then explains how to overcome the noise issues in an MCR using a secure microcontroller optimized for the task. One can certainly use Maxwell's field equations, the geometry of the MRH, and the boundary conditions to predict the MRH output-voltage behavior. However, this approach is complicated and provides limited insights for circuit analysis, design, and debugging. Instead, we propose characterizing the MRH first and then using basic circuit theory and a simple circuit simulator to analyze MCR behavior.

MAGNETIC-STRIPE CARD BASICS

Figure 1 shows a magnetic stripe card with three tracks. Several ISO/IEC standards define important card properties such as the physical size, exact location of the stripes, magnetic properties, and magnetic track data structures.[1] Track 1 standards were created by the International Air Transportation Association (IATA). Track 2 standards were created by the banking industry (American Bankers Association, ABA), and Track 3 standards were created by the thrift-savings industry.

A magnetic stripe card.

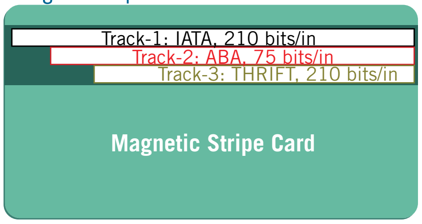


Figure 1

A two-frequency coherent phase (F2F) technique is used for encoding the data on magnetic stripe cards. As shown in **Figure 2**, the binary data is encoded along the track by magnetizing stripe areas with different polarities. The polarity of the transitions is arbitrary, since only the *relative space* between the transitions implies a binary 1 or a binary 0.

A binary 0 is encoded with a two-unit bar magnet, while a pair of one-unit bars represents a binary 1. Each bit occupies the same physical length on the stripe. A bit with an additional flux transition in the middle of its length is a binary 1.

The spectrum of a continuous signal with F2F coding contains two fundamental frequencies, f_0 and f_1, where f_0 is the fundamental of the square wave for binary 0 and $f_1 = 2f_0$ is the fundamental of the square wave for binary 1, hence the name F2F. The average amplitude of the binary 0 waveform is twice that of the binary 1 waveform, $A_0 = 2A_1$.

Figure 3 shows the combined spectrum of F2F encoded binary 0s and 1s normalized to f_0. Note that most of the signal energy resides between f_0 and 3 f_0. Thus, to get a good approximation of a rectangular waveform containing a series of F2F encoded binary 1s and 0s, it is enough to recover the two fundamentals (f_0 and f_1) and the 3rd harmonic of f_0.

Moreover, due to varying binary patterns there will be other components below f_0. However, we can see from Fourier analysis that the amplitudes of these components decrease quickly for decreasing frequencies. Thus, a bandwidth from $0.5f_0$ to $3f_0$ is adequate for recovering an F2F-encoded rectangular waveform.[2]

To estimate the minimum and maximum bit rates for f_0 and f_1, we need to know the swipe speed range and the track recording density. From ISO/IEC standards, the recording density of Tracks 1 and 3 is 210 bits/in (8.27 bits/mm), while that of Track 2 is 75 bits/in (2.95 bits/mm).

For calculating $f_{0,min}$ we take the slowest swipe speed supported and multiply it by the Track 2 density numbers. For $f_{1,max}$ we pick the fastest swipe speed supported and multiply it by the density of Tracks 1 or 3.

For target swipe rates of 2in/s to 100 in/s (5cm/s to 254cm/s), the range of f_0 and f_1 values is calculated as:

Tracks 1 and 3: $f_{0,min} = 0.42$kbps and $f_{1,max} = 42$kbps
Track 2: $f_{0,min} = 0.15$kbps and $f_{1,max} = 15$kbps

The importance of knowing $f_{0,min}$ and $f_{1,max}$ will become clear once we have the MRH model and study its transfer function.

FUNDAMENTALS OF MAGNETIC READ HEADS AND CARD READERS

Swiping a magnetic stripe card past a stationary MRH results in a changing magnetic flux that produces a moving electric field. Thus, a voltage is induced at the MRH's output. Figure 2 can be used to study the reading process. Starting from the top, a magnetic stripe moving past an MRH results in changing flux events that induce a voltage at the MRH's output terminals. The open-circuit readback voltage without any electrical losses is given by the well-known expression:[3]

$$E(\overline{x}) = K\frac{d}{dt}\int_{y_1}^{y_2}\cdot\int_{-\infty}^{+\infty} H(x,y)\cdot M\left[\left(x-\overline{x}\right),y\right]dx\,dy$$

Where:

- $E(\overline{x})$ = open-circuit voltage.
- K = a constant relating the effects of magnetic-stripe velocity, head width, and the number of coil turns in the MRH.
- $H(x, y)$ = field function of the read head.
- $M[(x-\overline{x}), y]$ = magnetic-stripe material magnetization.
- y_1 = spacing from the head to the top of the magnetic-stripe.
- y_2 = spacing from the head to the bottom of the magnetic-stripe.

F2F encoding and decoding waveforms.

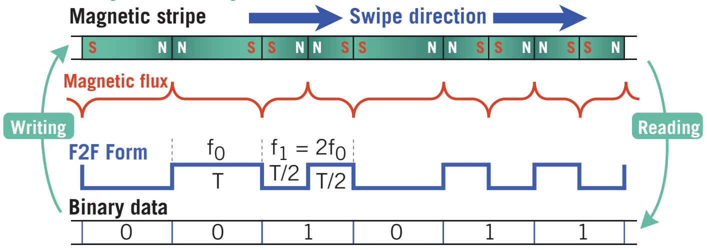


Figure 2

Spectrum of F2F-encoded binary 0 and 1.

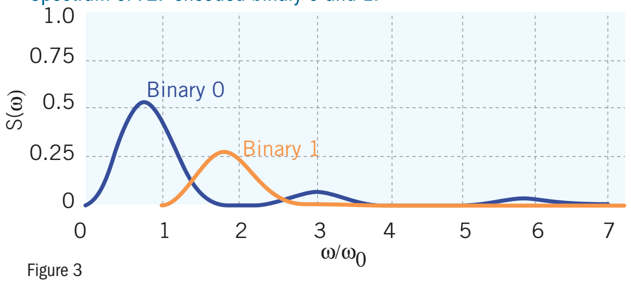


Figure 3

It all Rides On the Right Technology

Get Pattern Recognition Right with CogniMem!™

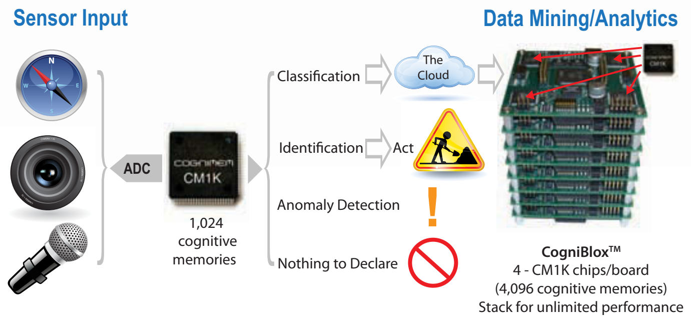
Efficient pattern recognition and data mining demands a departure from traditional computing technology and programming. What the wheel did to revolutionize transportation, CogniMem does for pattern recognition.

Native Parallelism for High-Performance Reduced-Energy Computing

Match a pattern among any number in a fixed amount of time. Forget about the memory size and access-time bottlenecks. Take advantage of the natively parallel architecture of our cognitive memories for massively parallel data mining. Seamlessly stack CogniBlox™ for scalable high-performance recognition with reduced energy, latency, size and cost.

Native Model Generator and Real-Time Classifier for Intelligent Sensing

Turn digital input streams into valuable information at the sensor level whether it is to actuate a device, warn for anomalies or send only relevant information to the cloud. Our cognitive memories learn in real-time from examples which eliminates programming and improves accuracy.



Get Your Pattern Recognition Rolling with CogniMem

Eval Kit/SDK
- CM1K + vision sensor
- Freescale i.MX53 Quick Start Board
- ~~$1,894~~ ***$995*** while supply lasts

Technical Training
- Learn about pattern recognition from the industry pioneers in on-line and classroom training.

COGNIMEM
Technologies, Inc.
(916) 358-9483 • www.cognimem.com

Clearly, this equation is highly complicated and not intuitive for circuit analysis and design, but we can use the basic principles to determine a model as follows:

The MRH transfers magnetic energy to electric energy. Since the MRH's input is a changing magnetic field and its output is a changing electric field, the model should contain at least one inductive element, L_h, and one capacitive element, C_h. In real systems some energy is always spent during the transformation. Thus, the model must also contain a resistive element, R_h. In practice, an MRH will not only have C_h across its two terminals, but it will also have an external impedance, Z_O. from, for example, connecting wires, PCB traces, IC pins, probes, etc.

A model corresponding to the *n*th harmonic frequency, f_n, is shown in **Figure 4a**, which then is simplified as **Figure 4b.**[3,4,5] The transfer function of the 2nd-order circuit in Figure 4b is easily calculated as:

$$Tn(s) = \frac{V_n(s)}{E_n(s)} = \frac{\dfrac{1}{C_o L_h(f_n)}}{s^2 + s\left[\dfrac{R_h(f_n)}{L_h(f_n)} + \dfrac{1}{R_o\, C_o}\right] + \dfrac{1}{L_h(f_n) C_o}\left[\dfrac{R_h(f_n)}{R_o} + 1\right]}$$

Note that the above transfer function does not contain any mechanical or magnetic terms, e.g., the swipe speed, head geometry, head and stripe separation, or the stripe magnetic properties. Thus, the transfer function is more intuitive for circuit design.

To further simplify things, we propose using a lumped model characterized at the highest system frequency instead of limiting the model to the swipe speeds.

Next we compare the electrical specifications of MRHs from some leading manufacturers. **Table 1** lists the key specifications that are needed for the model of Figure 4b. Notice the different amount of detail. While both Manufacturers A and C specify several electrical parameters, Manufacturer B specifies only one: peak-to-peak head readout level.

The following questions may arise about the missing information.

- **Head inductance (L_h).** How does L_h behave over a larger frequency range? How does L_h behave when carrying currents other than what is specified?
- **Head DC resistance (R_h).** What voltage level is applied across the head terminals?
- **Head read output level ($V_{O(P-P)}$).** What type of test card is used? What is the card swipe speed? What is the load across the head?
- **Head capacitance (C_h).** What is the capacitance between the two head terminals? Does it change with frequency?

To appreciate the importance of the above parameters, we examine the transfer function's denominator and find its roots by setting it to zero:

$$s^2 + s\left[\frac{R_h(f_n)}{L_h(f_n)} + \frac{1}{R_o C_o}\right] + \frac{1}{L_h(f_n)C_o}\left[\frac{R_h(f_n)}{R_o} + 1\right] = 0$$

To keep expressions simple and better understand the second-order behavior, several network analysis books use standard forms to write equations.[6,7] One standard uses the form:

$$s^2 + 2\alpha \cdot s + \omega_o^2 = 0$$

whose roots are:

$$s_1, s_2 = -\alpha \pm \sqrt{\alpha^2 - \omega_o^2}$$

Where α is the damping attenuation:

$$\alpha = \left(\frac{R_h}{2L_h} + \frac{1}{2R_o \cdot C}\right)$$

ω_o is the resonance frequency:

$$\omega_o = \frac{1}{\sqrt{L_h \cdot C}} \sqrt{\left(\frac{R_h}{R_o} + 1\right)}$$

Therefore, depending on the values of α and ω_o, the roots of the natural response can be *real*, *complex*, or *imaginary*. For readers who are familiar with other standard forms, we now define the damping factor as $\zeta = \alpha/\omega_o$ (note: quality factor $Q = 1/2\zeta$) and use the other standard form:

$$s^2 + 2\zeta\omega_o \cdot s + \omega_o^2 = 0$$

whose roots are:

$$s_1, s_2 = -\zeta\omega_o \pm \omega_o\sqrt{\zeta^2 - 1}$$

When a nonzero forcing function (such as a step, a ramp, or an impulse) is applied to the system, the location of the roots in the s-plane directly affects the settling behavior. **Figure 5** shows the settling behavior for various ζ values when a step is applied at $t = 0$. Specifically, the settling behavior is categorized as:

$\zeta > 1 \rightarrow$ *overdamped*
$\zeta < 1 \rightarrow$ *underdamped*
$\zeta = 1 \rightarrow$ *critically damped*
$\zeta = 0 \rightarrow$ *undamped* or *oscillatory*

From Figure 5 we observe that in an *underdamped* system, ringing occurs that can cause reading errors due to false peaks and false zero crossings. However, if the system is drastically *overdamped*, timing errors can occur from slow settling and reading errors can occur from shifts in the peaks. After analyzing an MRH's time-domain behavior, we next look at its frequency-domain behavior.

Equivalent MRH model.

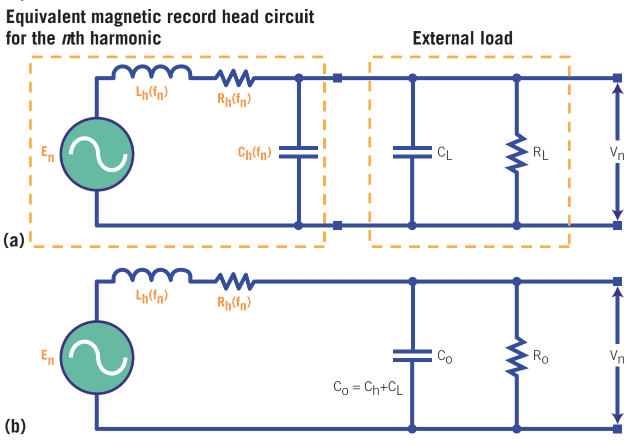


Figure 4

Manufacturer specifications for a magnetic head.

Parameter	Manufacturer		
	A	B	C
$V_{O,P-P}$ (mV)	20	19	35
L_h (mH)	25	–	110
C_h (pF)	–	–	–
R_h (Ω)	110	–	280

Table 1

MRH output voltage for various ζ values.

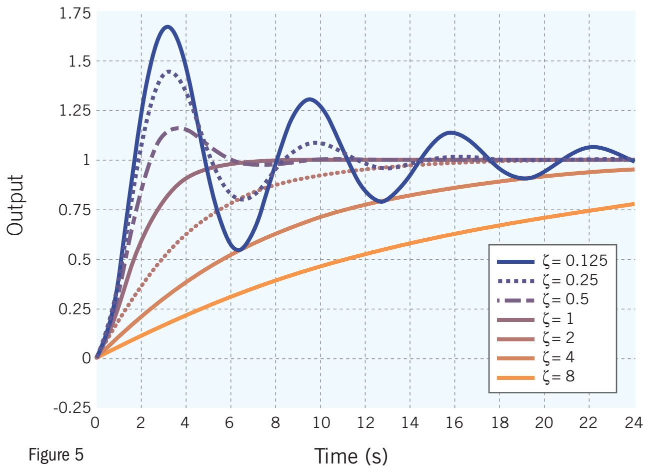


Figure 5

Transfer function's frequency response.

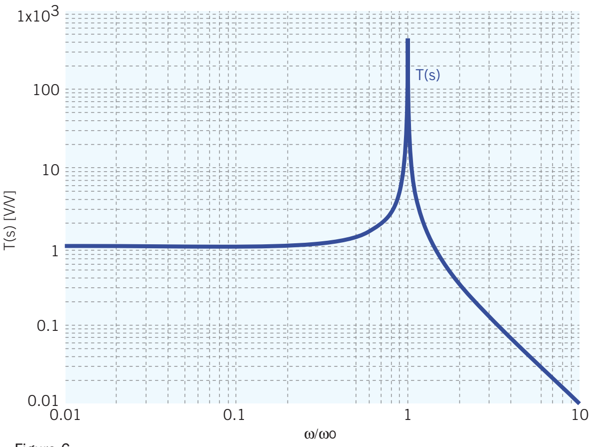


Figure 6

Distortion in MRH output voltage due to gain peaking.

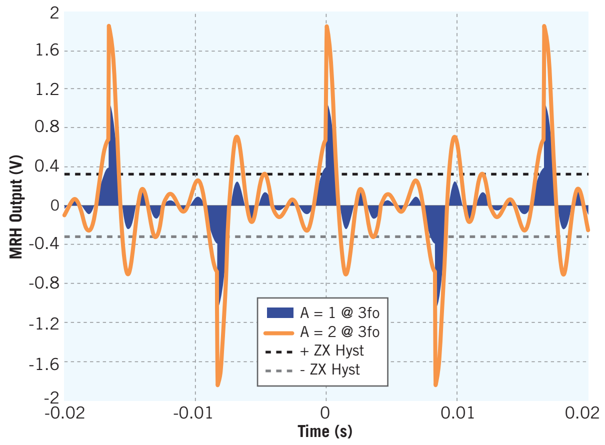


Figure 7

Figure 6 shows the frequency response of the transfer function, $T_n(s)$, which is normalized to its resonance frequency, ω_o. We observe peaking as we approach the system resonance frequency. This is due to the intrinsic nature of the circuit shown in Figure 4b, in other words, a parallel RLC. Depending on the swipe speed, this peaking can also cause reading errors.

Recall that when trying to recover F2F-encoded binary data, we need the two fundamental frequencies, f_0 and f_1, and at least the 3rd harmonic of f_0. From Figure 3 we see that most of the signal energy is in the vicinity of $0.5f_0$ to $3.5f_0$, while a small portion is around $6f_0$.

What will happen to the recovered F2F waveform if higher harmonics were gained up? From Fourier analysis we know that the recovered waveform shape will change as the magnitudes of the higher harmonic coefficients change. Thus, for some cases the gain peaking shown in Figure 5 can reach a point where the 3rd and the 6th harmonics are amplified to such a level that the recovered signal is severely distorted. Any distortion that results in false peaks and zero crossings will lead to reading errors.

The above point is emphasized in **Figure 7**, which mathematically shows MRH output voltage for two different gains at higher harmonics: solid blue for the unity gain and orange for a gain of two. The dotted black lines are the zero crossing detector hysteresis limits. Clearly, as the gain doubles, the MRH output signal in red shows more distortion, false peaks, and zero crossings.

Furthermore, the MRH will not only experience disturbances due to varying card-swipe speeds, it will also see disturbances at much higher frequencies that might be present in the overall system, for example, a high-frequency system clock. Because of gain peaking, this too can result in signal distortion and possible reading errors. Thus, for designing an optimum card-reading system, it is crucial to know the frequency behavior of the MRH beyond the swipe rates. One must characterize the MRH frequency behavior at least to the highest system frequency.

CHARACTERIZING VARIOUS READ HEADS

We used a commercially available impedance/gain-phase analyzer to find the equivalent circuits of several MRHs from different manufacturers. The characterization included single-, double-, and triple-track MRHs that were used in the MCR based on a MAXQ1740 secure microcontroller. As 12MHz is the maximum system clock frequency for the MAXQ1740, each MRH was characterized from 100Hz to 12MHz (100Hz is analyzer's limit). **Table 2** shows parameter averages for triple-track MRHs.

CKT-B parameters for triple-track MRHs.

Parameter	MRH-1	MRH-2	MRH-3	MRH-4
L_h (mH)	13.67	58.09	13.20	57.43
C_h (pF)	22.15	31.11	20.60	16.97
R_h (Ω)	146.78	234.57	145.72	214.51

Table 2

MRH transfer function vs. frequency with a 1GΩ external load (ζ = 0.03).

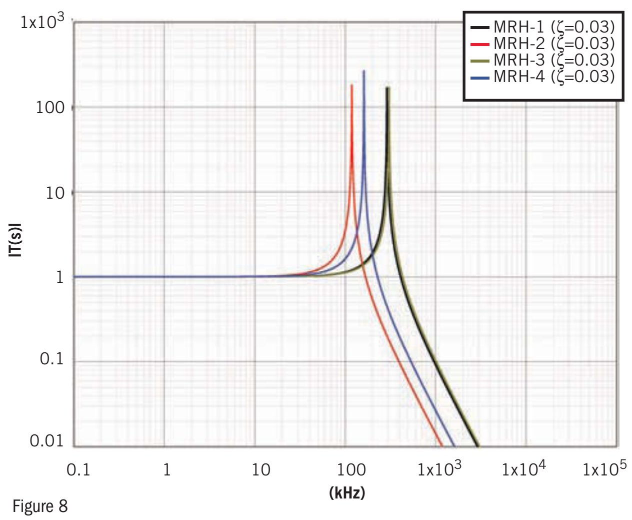


Figure 8

MRH transfer function vs. 3rd and 6th harmonic frequency range .

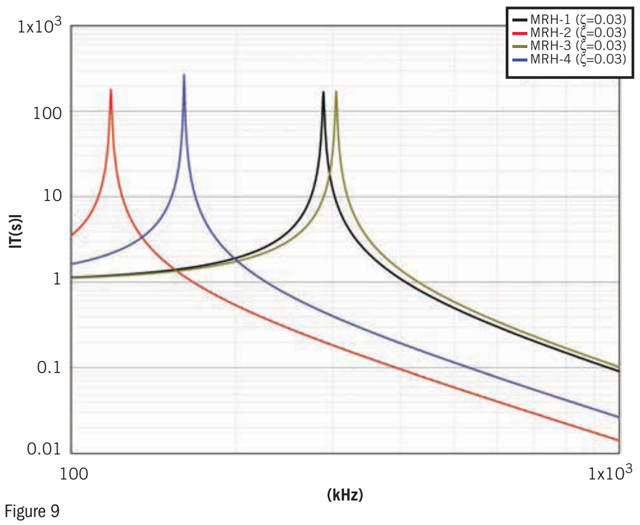


Figure 9

ANALYZING THE MEASURED PARAMETERS

Comparing the parameters in Table 2, we see that MRH 1 and MRH 3 are similar. The relative differences between their parameters are $\Delta L_h \sim 3.6\%$, $\Delta R_h \sim 0.7\%$, and $\Delta C_h \sim 7.5\%$. For MRH 2 and MRH 4, the relative differences in their parameters are $\Delta L_h \sim 1.2\%$, $\Delta R_h \sim 9.4\%$, and $\Delta C_h \sim 83\%$.

Since C_h affects both α and ω_o, for similar conditions we can expect the behavior of MRHs 1 and 3 to be similar. We can expect the behavior of MRHs 2 and 4 to track below their resonance frequencies, but then change as the frequencies reach close to, and beyond, their respective resonance points.

The last two points become obvious when we plot the transfer function's frequency response for the characterized MRHs as shown in **Figure 8**. The load in Figure 8 is 1G, which results in a damping ratio of 0.03. The plots for MRHs 1 and 3 are virtually the same, while the plots for MRHs 2 and 4 show increasing differences around the resonance frequencies. The increase in magnitude could result in reading errors, as described earlier.

Figure 9 shows MRH transfer functions for the frequency range of 150kHz to 300kHz, i.e., 3rd and 6th harmonics corresponding to the maximum card-swipe rate of 100in/s (254cm/s). We can see that as the swipe rates increase, so do the MRH transfer-function magnitude values. The main concern here is that if higher harmonics are gained up beyond a point, false zero crossings and peaks can occur, as shown in Figure 7. Also, if signals larger than the maximum allowed appear at the interface between the head and the card reader's inputs, then reading errors can occur.

Two factors cause a larger signal. First, a faster swipe increases the rate of magnetic flux change. This, according to Faraday's law, induces a larger inductor voltage, resulting in a larger inductor current. Second, a larger current flowing through a larger impedance results in a larger output voltage, in accordance with Ohm's law.

Within the card swipe-rate range we need to limit the peaking to less than or equal to 20, which is the ratio between the maximum and minimum gains of the MAXQ1740 magnetic card reader. Figure 9 shows that the impedance change for MRHs 1 and 3 is less than 20, but the change for MRH 2 and MRH 4 is nearly 30 and exceeds the limit of 20.

What will happen if a damping resistor is added at the output of the MRH? **Figure 10** shows the transfer function plots for three arbitrarily different external loads values: 100kΩ, 10kΩ, and 1kΩ. We see in Figure 10 that for lower-value external resistors, the peaking is reduced compared to that shown in Figure 9. Note that for 1kΩ loads the gain at the 3rd harmonic is severely reduced for all four MRHs. This can be a problem. With 100k loads, for MRHs 2 and 3, the gains peaks at the 3rd harmonic, while for MRHs 2 and 3, the gain peaks at the 6th harmonic. The main point is

MRH transfer function for different external load values.

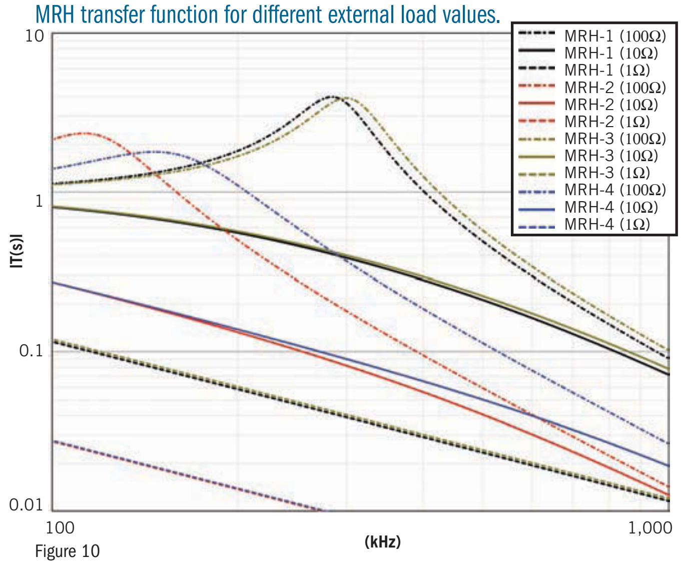


Figure 10

MRH damping factors(ζ) vs. external resistor(R_0).

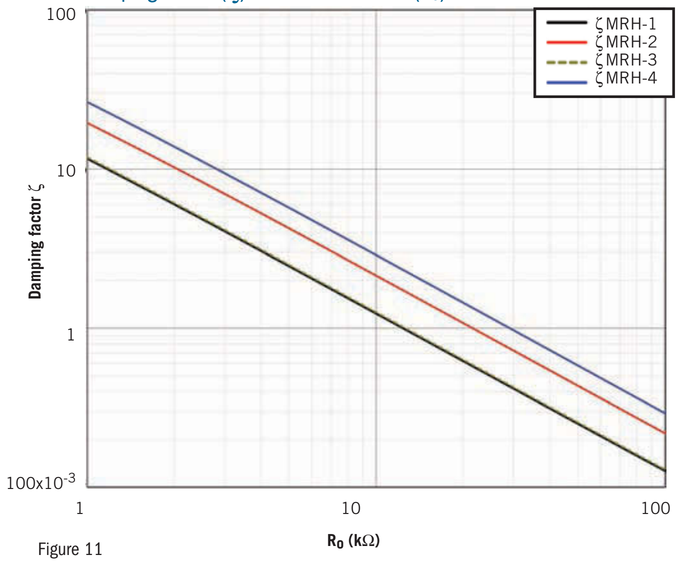


Figure 11

MRH transfer functions for optimal external loads.

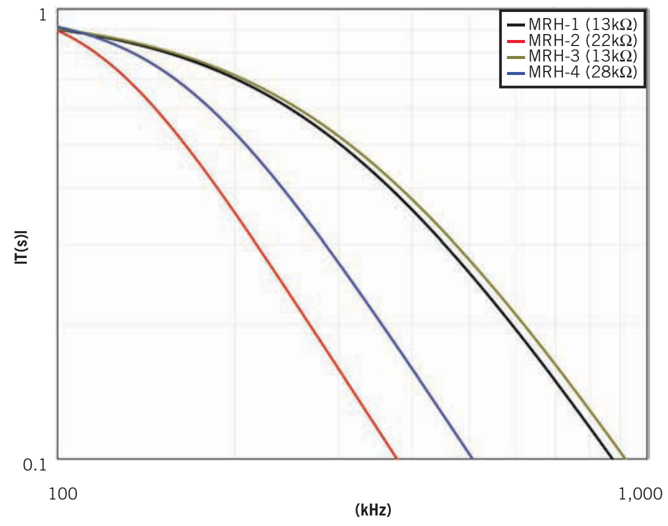


Figure 12

that we cannot arbitrarily pick the R_o values.

When using external resistor R_o across the MRH terminals, it's important to ensure that the damping ratio, ζ, stays as close to the unity as possible. **Figure 11** plots ζ vs. R_o for the four characterized MRHs. For ζ=1, we need $R_o \approx$ 12kΩ for both MRHs 1 and 3; $R_o \approx$ 22kΩ for MRH 2; and $R_o \approx$ 28kΩ for MRH 4. **Figure 12** shows that transfer function with optimum load values. Comparing Figure 12 to Figure 10, we note that the gain does not peak at the 3rd harmonic and stays close to unity.

While the maximum R_o is set by ζ=1, the minimum R_o value depends on the minimum signal supported and the head DC resistance, R_h. As a general rule, keep $R_o \geq 5R_h$ so that R_o in parallel with R_h will not attenuate the head output signal by more than 20%.

There are several key points to remember. First, due to parallel RLC, the transfer function peaks around the resonance frequency, ω_o. Therefore, limit this peaking for the range corresponding to the 3rd and the 6th harmonics of card swipe rate, e.g., 150kH to 300kHz for the card swipe-rate range of 42kH to 50kHz. Second, the system behavior can be adjusted by placing R_o across the head reader's terminals. Changing R_o changes the damping ratio, ζ. Finally, select R_o value to make the system critically damped and let the lead wiring and PCB routing set the C_o value.

OPTIMIZING CARD READING

With a method now to resolve the discrepancies in MRH specifications, we can improve card reading performance. Our focus will be on reducing the effects of noise, which mostly affect the zero crossings (ZX). After characterizing the MRH model for the entire frequency range (note: the characterization must include lead wires and the PCB routing), we follow these steps:

If using less expensive but noisier read heads, overcome the noise by reducing the input signal without affecting the damping ratio.

Step 1. Choose an R_o value to get an appropriate damping ratio and limit the gain peaking.

- In general, the target should be critically damped to slightly overdamped. As an exception, if for some cases the gain at the 3rd harmonic drops below half, we can equalize the gain by a slightly underdamped system.
- An underdamped system can introduce noise from ringing of the input signal. Ringing noise adversely affects the zero crossing detector, but may also result in false peaks due to gain peaking.
- Keep $R_o \geq 5R_h$ with the maximum R_o set by ζ =1.

Step 2. On noisier printed circuit boards (PCBs) it helps to make the system overdamped, especially Track 2 (T2).

- T2 has 40 numeric digits, as opposed to 79 alphanumeric characters for T1/T3.
- On T2 longer gaps exist between the peaks where noise can affect ZX.
- Overdamping integrates the T2 signal. The signal approaches a sawtooth waveform, as shown in **Figure 13**. Overdampening helps the ZX by filtering out high-frequency glitches.
- Keep $R_o \geq 5R_h$ so that the head attenuation stays under 20%.
- A note of caution: an excessively overdamped system can lead to errors due to slow settling and peak shifts.

Step 3. If using less expensive but noisier read heads, overcome the noise by reducing the input signal without affecting the damping ratio.

- Choose an appropriate R_o.
- Divide R_o into smaller segments so that the total R_o remains the same as in Step 1.
- Use the appropriate tap to get the required signal division.

Overdamped response. T2 for a manual swipe with 40% card and R_o =1.5kΩ.

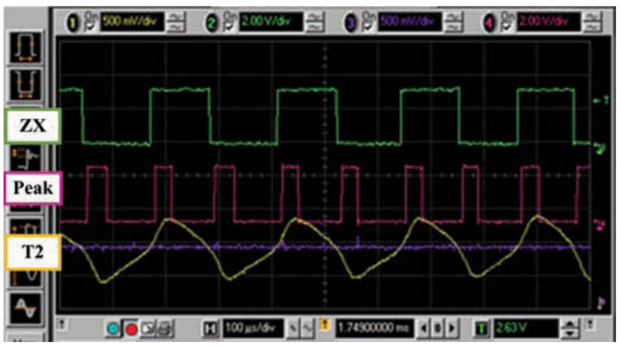


Figure 13

Critically damped behavior. T2 for a manual swipe with 40% card and R_o =13.5kΩ.

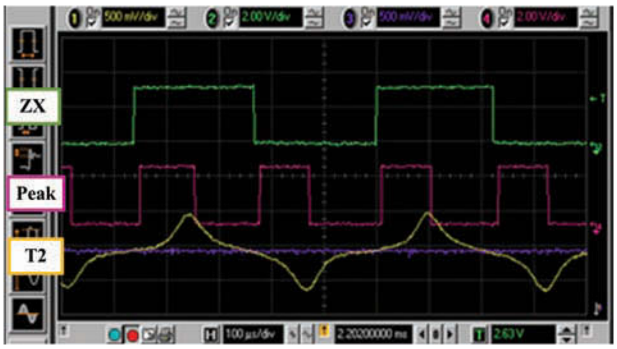


Figure 14

- Several ways to do this are described under Practical Examples below.

Step 4. When the read head output on the MAXQ1740 exceeds 300mV$_{P-P}$, internal clipping of the signal occurs. This clipping can also cause reading errors.

- Use the method described in Step 3 to reduce the signal.

PRACTICAL EXAMPLES

Input signal and noise reduction

Suppose the optimized output resistor value is R_o.

Goal: achieve a 25% reduction in the signal.

- Use one 0.25 R_o and one 0.75 R_o in series across the head. Then 0.75 R_o is tied to the head common-pin side. Tie the midpoint to the input.
- Use four 0.25 R_o in series across the head. Tie the midpoint to the input.

Goal: achieve a 75% reduction in the signal.

- Use one 0.25 R_o and one 0.75 R_o in series across the head. Then 0.25 R_o is tied to the head common-pin side. Tie the midpoint to the input.
- Use four 0.25 R_o in series across the head. Tie one tap above midpoint to the input.

EFFECTS OF DAMPING FACTORS

We next consider various damping factors and their effects on the actual signal behavior when test magnetic cards are swiped using an MCR based on the MAXQ1740. MRH 2 was used in the tests. There are two important things to note about the test cards used. First, cards are commercially available from Q-Card and follow ISO/IEC 7811 through 7816 standards. Second, the card signal amplitudes are specified as a percentage of the nominal level. Thus, a 40% card implies a maximum output level that is 40% of the nominal ISO level.

Before we had the MRH model available, it took time-consuming and frustrating guesswork to determine the right value of external resistors. With the model available here, we solved the noise issues by using a 13.5kΩ external resistor that matched our model prediction. Figure 13 shows the overdamped behavior, while **Figure 14** shows the critically damped behavior. Comparing Figures 11 and 12, we note the slow settling and peak shifts for the overdamped case compared to the critically damped behavior. Both slow settling and peak shifts can cause timing errors resulting in reading errors as described earlier.

With the model available here, we solved the noise issues by using a 13.5kΩ external resistor that matched our model prediction.

TAKE OUT DA NOISE

Using methods presented in this article, one can predict and prevent potential problems early in the design phase or when deciding which MRH to pick. For example, designers can now anticipate that in an underdamped system, reading errors can occur due to false peaks and false zero crossings. Both ringing and excessive gain peaking (around the 3rd and 5th harmonics of the swipe speed) can produce false peaks and zero crossing. Conversely, if the system is drastically overdamped, timing errors can occur because of peak shifts.

The methods presented here are also useful for improving the performance of an existing card-reader system that uses a specific read head. For example, in a noisy system one can first use several series external resistors to make the system critically damped and then tap the MRH output from an appropriate node to divide down the MRH output level. Finally, the methods were verified in an actual card-reader system based on the MAXQ1740 microcontroller. ■

Irfan A. Chaudhry was a principal member of the technical staff for IC design at Maxim Integrated Products at the time this article was written but is no longer with the company. He joined Maxim in 2009 with over 16 years of mixed-mode IC design experience in data converters, hard disk drive controllers, nuclear weapon testers, and power management. He has more than two dozen designs in production and holds four U.S. patents. He attended the University of Idaho and Washington State University for his undergraduate and graduate studies, respectively.

ACKNOWLEDGEMENTS

Our methods are the result of rediscovering the elegant methods developed by A. S. Hoagland in the late 1950s and published in his 1963 book.[3]

Many thanks are due to Maxim engineers: Steve Grider who provided the most help; Bryan Taylor, Stephen Umfleet, and Lonnie Hornsby for their help in the lab; Kevin Kwak, Don Pearce, Mark Weldele, Gary Zanders, Brandon Priddy, Nadeem Mirza, Mark Lovell, Kathy Vehorn, Aaron Minor, and Jeff Owens for their design, test, and system-level contributions and suggestions.

ENDNOTES

1. ISO/IEC 7810, ISO/IEC 7811, ISO/IEC 7812, ISO/IEC 7813. *www.iso.org/iso/search.htm?qt=identification+cards&searchSubmit=Search&sort=rel&type=simple&published=on.*
2. Cuccia, C. L. *Harmonics, Sidebands and Transients in Communication Engineering.* McGraw-Hill, New York, 1952.
3. Hoagland, Albert. *Digital Magnetic Recording*, Wiley, New York, 1963.
4. Chu, W. W. "Computer Simulations of Waveform Distortions in Digital Magnetic Recordings," *IEEE Transactions on Electronic Computers*, Vol. 15, pp. 328–336, Jun. 1966.
5. Chu, W. W. "A Computer Simulation of Electrical Loss and Loading Effect in Magnetic Recording," *IEEE Transactions on Electronic Computers*, Vol. EC-16, No. 4, pp. 430–434, Aug. 1967.
6. Nilsson, J.W. *Electric Circuits*, 3rd ed., (Reading, MA, Addison-Wesley Publishing Co.), 1990.
7. VanValkenger, M.E. *Network Analysis,* 3rd ed., (Englewood Cliffs, NJ Prentice-Hall), 1974.

By Jack G. Ganssle

Probing pointers, take 2

Last month I gave a mostly theoretical overview of the effect probes—like scope and logic analyzer probes—have on the nodes being tested. The most important effects stem from the capacitance of the probe tip. To reiterate, the reactance, or resistance to AC, at the tip is:

$$X_c = \frac{1}{2\pi fC}$$

This reactance loads the node and can alter a device's operation—or worse.

To explore this, I built a circuit on a printed circuit board with ground and power planes, keeping all wires very short. A 50-MHz oscillator drives two AND gates. The 74AUC08 is spec'd with a propagation delay between 0.2 and 1.6 nsec at the 2.5 volts I used for the experiment. The second gate is a slower 74LVC08 whose propagation delay is 0.7 to 4.4 nsec. Still speedy, but slower than the first gate. I was not able to find rise-time specifications but assumed the faster AUC would switch with more alacrity and thought it would be interesting to compare effects with differing rise times. Alas, it was not to be; the LVC wasn't much slower than the AUC. So I'll generally report on the slower gate's results

These parts are in miniscule SOT-23 packages, which keeps inductances very low but means one solders under a microscope, sans coffee.

I wanted to see the effect that probes have on nodes, but that posed a meta-problem: if probing causes distortion, how can one see the undistorted signal? Thankfully there's a simple solution. I made a pair of meter-long probes from RG-58/U coax cable. A BNC connector on one end goes to the scope. A short bit of braid is exposed and soldered to the ground plane very close to the node being probed, and a ¼-watt 1K resistor goes from the inner conductor to the node. I used an Agilent MSO-X-3054A scope with selectable input impedance, set to 50 ohms. This is critical for the shop-made probe; the normal 1 MΩ simply will not work. If your scope doesn't have a 50-Ω mode, use a series attenuator such as the 120082 from Test Products International (this part doesn't seem to be on their web page, but Digikey resells them). Agilent's N5442A is a more expensive but better-quality alternative.

Jack tests several probes to see how different probes change the results.

RG/58U is 50-Ω cable; add the resistor and the total is 1,050 ohms. The scope's 50-Ω input forms a 21:1 divider, but the resistor's very low capacitance (remember, a ¼-watt resistor runs only 0.5 pf) means the probe's tip looks extremely resistive, with little reactance. The scope thinks a 1X probe is installed, so to accommodate the oddball 21:1 ratio one multiplies the displayed readings by 21.

The first experiment showed Fourier at work. The blue trace in **Figure 1** shows the output of the fastest gate using a 21X probe. Note that it's far from perfect since the circuit had its own reactive properties. The rise time (measured with a faster sweep rate than shown) is about 690 psec (picoseconds). "About" is the operative word, as the scope has a 500-MHz bandwidth (though samples at 4 GS/sec). I found that having the instrument average readings over 128 samples gave very consistent results.

The pink trace is the Fourier Transform of the gate's output. Unlike the blue trace, this one is not in the time domain (e.g., time across the horizontal axis) but is in the frequency domain. From left to right spans 2 GHz, with 500 MHz at the center. The vertical axis is dBm, so is a log scale. Each peak corresponds to a term in the Fourier series. Point "A" is exactly 50 MHz, the frequency of the oscillator. Most of the en-

Jack G. Ganssle is a lecturer and consultant on embedded development issues. He conducts seminars on embedded systems and helps companies with their embedded challenges. Contact him at jack@ganssle.com.

ergy is concentrated there. Peak "B" is 48 dBm down from "A." That's on the order of 100,000 times lower than "A."

"B" is at 900 MHz. Remembering that little energy remains in frequencies above

$$F = \frac{0.5}{T_r}$$

with F=900 MHz the rise time is 555 psec, close enough to the 690 measured. The same experiment using the slower 74LVC08 gate yielded 48 dBm down at 450 MHz, or a rise time of 1.1 nsec. That's close to the 0.95 nsec reported by the scope.

Next, I connected a decent-quality $200 Agilent N2890A 500-MHz probe (11-pf tip capacitance) on the 74LVC08's output. The 21X probe saw an additional third of a nanosecond in rise time due to the N2890A's capacitance. In other words, connect a probe and the circuit's behavior changes.

In **Figure 2** the orange trace is the gate's output measured, as usual, with the 21X probe, although now there's 10 inches of wire dangling from it. That trace is stored as a reference, and the green one is the same point, with the same probe, but the N2890A is connected to the end of that 10 inches of wire. Note that the waveform has changed—even though that other probe is almost a foot away—and the signal is slightly delayed. This is probably not going to cause much trouble.

Rise time spiked to 5.5 nsec, more than a five times increase; I suggest immediately combing your lab for X1 probes and donating them to Goodwill.

Gates typically have a very low output impedance, so it's unsurprising there's so little effect. Often, though, we're sensing signals that go to more than one place. For instance, the "read" control line probably goes from the CPU to quite a few spots on the board. To explore this situation, I put the 21X probe five inches down that wire, captured the waveform into the reference (orange in **Figure 3**), and then connected the same N2890A at the end of the 10 inches of wire. The signal (green) at the 5-inch point shifted right and was distorted.

Consider the clock signal: On a typical board, it runs all over the place. The impedance at the driver is very low, but the long PCB track will have a varying reactance. Probe it and the distortion can be enough to cause the system to fail.

The ringing is caused by an impedance mismatch. The N2890A has changed the node's impedance, so it no longer matches that of the driver. Part of the signal is reflected back to the driver, and this reflection is the bounciness on the top and bottom of the pulses.

I didn't have any X1 probes around, so put a 100-pf capacitor on the node to

Fourier Transform of fast edges.

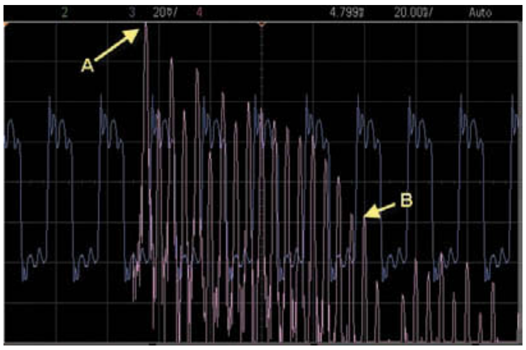


Figure 1

A node's signal changes when a probe is attached 10 inches away.

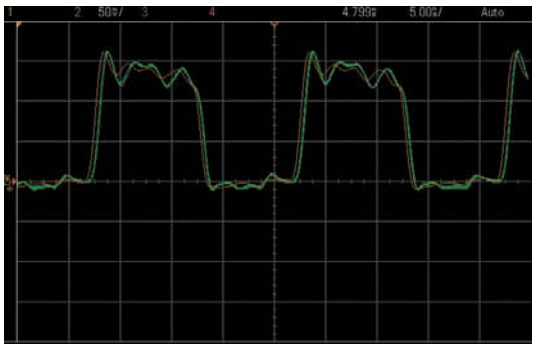


Figure 2

Distortion 5 inches down the wire due to a probe at the 10-inch point.

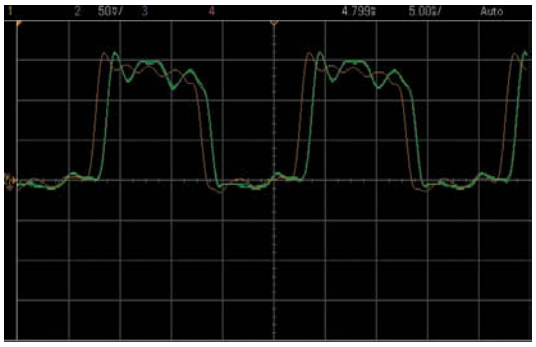


Figure 3

Distortion at the output of the gate with a 30-pf simulated probe.

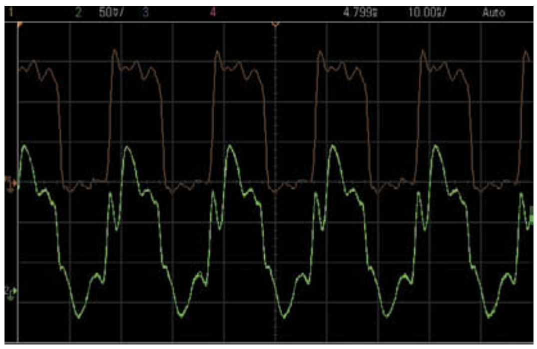


Figure 4

simulate a really crappy probe. Rise time spiked to 5.5 nsec, more than a five times increase, and the signal was delayed by almost a nsec. I suggest immediately combing your lab for X1 probes and donating them to Goodwill. And be very wary of ad hoc connections—like clip leads and soldered-in wires—whose properties you haven't profiled.

But 100 pf is a *really* crummy probe. I soldered a 30 pf cap on the node to simulate one that's somewhat like an ad hoc connection or a moderately-cheap probe. In **Figure 4**, the orange trace is the gate's output with no load—just the 21X probe. The green is with the additional 30 pf. The distortion is significant.

So a 30-pf probe grossly reshapes the node's signal. What effects could that cause?

First, everything this signal goes to will see a corrupt input. If it goes to a flip flop's clock input the altered rise time could cause data to be incorrectly latched. Or, if the flop's data input(s) are changing at roughly the same time, the flop's output could become metastable —it'll oscillate for a short time and then settle to a random value.

If it goes to a processor's non-maskable interrupt input the leading-edge bounce could cause the CPU to execute two or more interrupts rather than one. (Generally this is not a problem for normal maskable interrupts since the first one disables any others).

But wait, there's more. Note that the signal extends from well below ground (about -600 mV) to 3.7 volts (be sure to factor in the attenuation of the 21X probe), which is much higher than the 2.5-volt Vcc. Depending on the logic family this signal goes to, those values could exceed the absolute maximum ratings. It's possible the driven device will go into SCR latchup, where it internally tries to connect power to ground, destroying the device. I have seen this happen: the chips explode. Really. It's cool.

The driven device will go into SCR latchup, where it internally tries to connect power to ground, destroying the device. The chips explode. It's cool.

So far I haven't shown any signals acquired by the N2890A. The yellow trace in **Figure 5**, is the gate's output using that probe. It's pretty ugly! The distortion is entirely in the probe, and not on the board, so does not represent the signal's true shape. In this case the probe is grounded using the normal 3-inch clip lead. Using the formula from last month, that loop has 61 nH of inductance.

In orange the same signal is displayed, but in this case I removed the probe's grabber and connected a very short, about 5 mm, ground wire to the metal band that encircles the tip. The signal is still not displayed correctly—it extends below ground and has a total magnitude of about four volts, much more than the 2.5 Vcc. But the better grounding did clean up the shape. The point is that poor grounding can cause the scope to display waveforms that don't reflect the node's real state.

The N2890A's result, with proper and poor grounds.

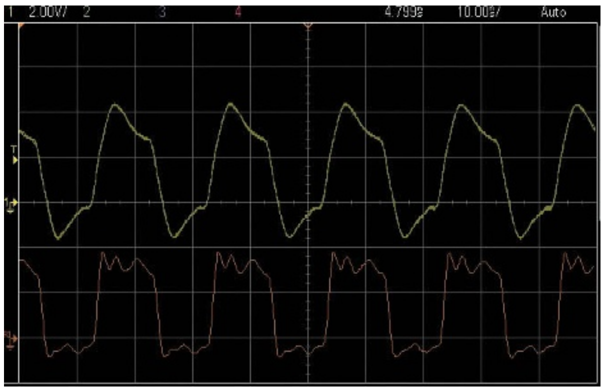


Figure 5

ELECTRONICS MATTERS

Many in the digital world find themselves divorced from electronics. We think in ones and zeroes, simple ideas that brook little subtlety. A one is a one, a zero a zero, and in between is a no-man's land as imponderable as the "space" that separates universes in the multiverse.

But electronics remains hugely important to digital people. Ignore it at your peril. Power supplies have crawled below a volt so the margin between a one and a zero is ever-tighter. On some parts the power supply must be held ±0.06 volts or the vendor makes no promises about correct operation. On a 74AUC08, typical fast logic, at 0.8 Vcc there's only a quarter volt between a high and a low. Improper probing can easily skew the node's behavior by that much. And, as we've seen, capacitance and inductance are so vital to digital engineering that we dare not ignore their effects when troubleshooting.

Reactance, impedance, and electromagnetics are big subjects that I've only lightly touched on. They're pretty interesting, too! I highly recommend the book *High-Speed Digital Design* for a deep and dirty look at working with high-speed systems.[1] *The ARRL Handbook* from the American Radio Relay League is possibly the best introduction to electronics available. It doesn't skimp on the math, but never goes beyond complex numbers. The focus is decidedly on radios, since this is the bible of ham radio, but the basics of electronics are covered here better than any other book I've found. There's a new edition every year; my dad bought me a copy in 1966, and since then I've "upgraded" every decade or so. ■

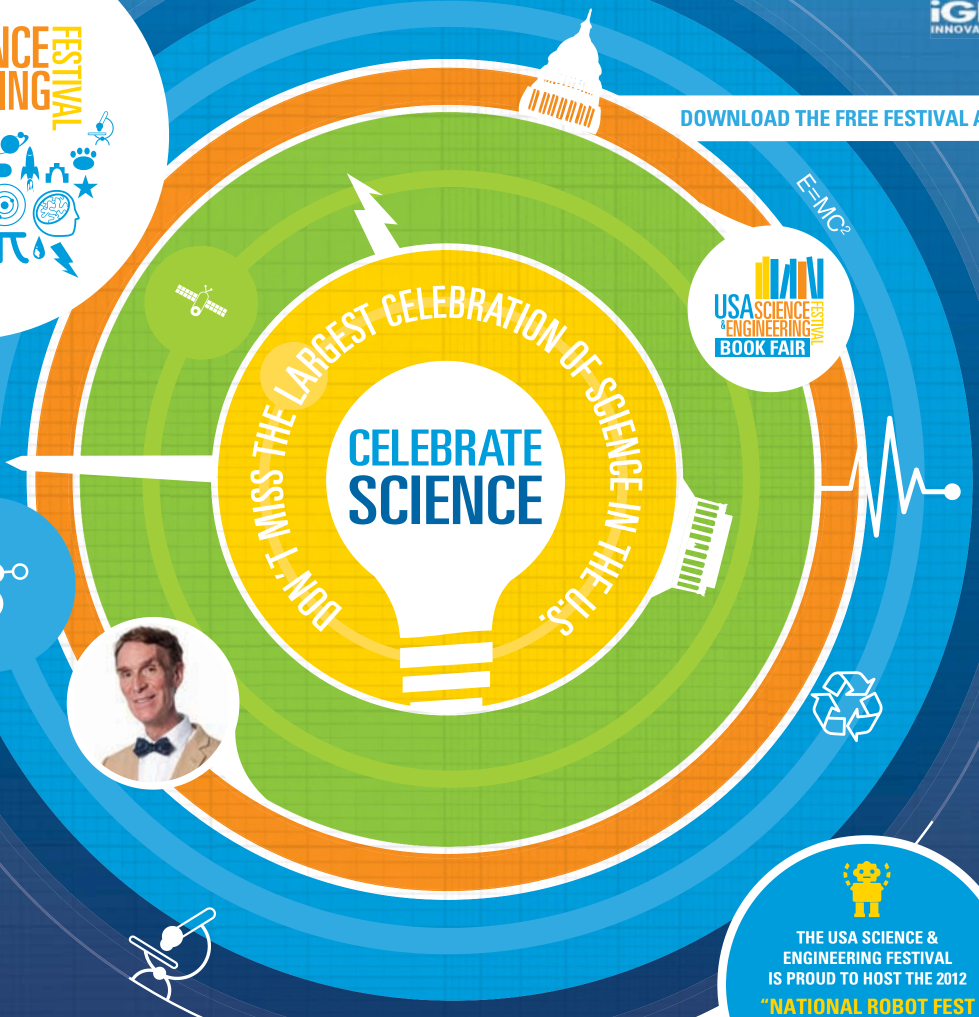
ENDNOTES

1. Johnson, Howard and Martin Graham. *High-Speed Digital Design* 1993 PTR Prentice-Hall Inc, Englewood Cliffs, NJ.
2. *The ARRL Handbook*, American Radio Relay League. Published afresh every year. *www.arrl.org.*

PROUD SPONSOR: EE Times iGEN INNOVATION GENERATION

DOWNLOAD THE FREE FESTIVAL APP (STARTING APRIL 10)

USA SCIENCE & ENGINEERING FESTIVAL



DON'T MISS THE LARGEST CELEBRATION OF SCIENCE IN THE U.S.
CELEBRATE SCIENCE
E=MC²
USA SCIENCE & ENGINEERING FESTIVAL BOOK FAIR

THE USA SCIENCE & ENGINEERING FESTIVAL IS PROUD TO HOST THE 2012 "NATIONAL ROBOT FEST AND DIY EXPO - WHERE CREATIVITY & TECHNOLOGY MEET"

USASCIENCEFESTIVAL.ORG

GRAND FINALE

EXPO & BOOK FAIR

APRIL 28 & 29, 2012

WALTER E. WASHINGTON CONVENTION CENTER, WASHINGTON, D.C.

OVER 3000 FUN HANDS-ON ACTIVITIES AND MORE THAN 100 STAGE SHOWS
MEET AWARD-WINNING AUTHORS AND SCIENCE CELEBRITIES LIKE BILL NYE THE SCIENCE GUY AND ADAM SAVAGE & JAMIE HYNEMAN
NEW THIS YEAR: CAREER PAVILION & BOOK FAIR | A FREE EVENT

LOCKHEEDIUM / FESTIVAL HOST

LOCKHEED MARTIN

BOCKIUM

Larry and Diane Bock

K&L GATESIUM

K&L GATES

NOBELIUM

NORTHROP GRUMMAN Foundation

SCIENTIFIC AMERICAN

POPULAR SCIENCE

NewScientist

EE Times iGEN

technology review Published by MIT

Popular Mechanics

U.S. AIR FORCE

wunderground.com

EINSTEINIUM

National Institutes of Health

ResMed Foundation

at&t

C&EN

DOW

life technologies FOUNDATION

Microsoft

Farrell Family FOUNDATION

NSF

Twixie join the revolution

The Epoch Times

ENGINEERING.COM

KRYPTON

LEGO Education, CrazyEngineers.com, The Kavli Foundation, Medimmune, Sigma Xi, Illumina Inc., American Scientist, Physics Today, Forbes/Wolfe, SchoolTube, Washington Family Magazine, National Aeronautics and Space Administration, The Planetary Society, PBS Kids, ABC7/WJLA-TV, WAMU 88.5 - American University Radio, The George Washington University - School of Engineering and Applied Science (SEAS)

PLATINUM

Vertex Pharmaceuticals, Celestron, Baxter International, American Nuclear Society, Association of Science-Technology Centers, Xconomy, You Can Do the Rubik's Cube, Amgen, U.S. Environmental Protection Agency, U.S. Department of Defense, Society for Maintenance and Reliability Professionals (SMRP), Johns Hopkins University Applied Physics Laboratory, Raytheon, Center for America, U.S. Army RDECOM and eCYBERMISSION, Purdue University, Project Lead The Way (PLTW), The Scripps Foundation for Science and the Environment, Northern Virginia Technology Council, 3M, Aldebaran Robotics Inc., Center for Biotechnology Education at Johns Hopkins University

THANKS TO OUR SPONSORS

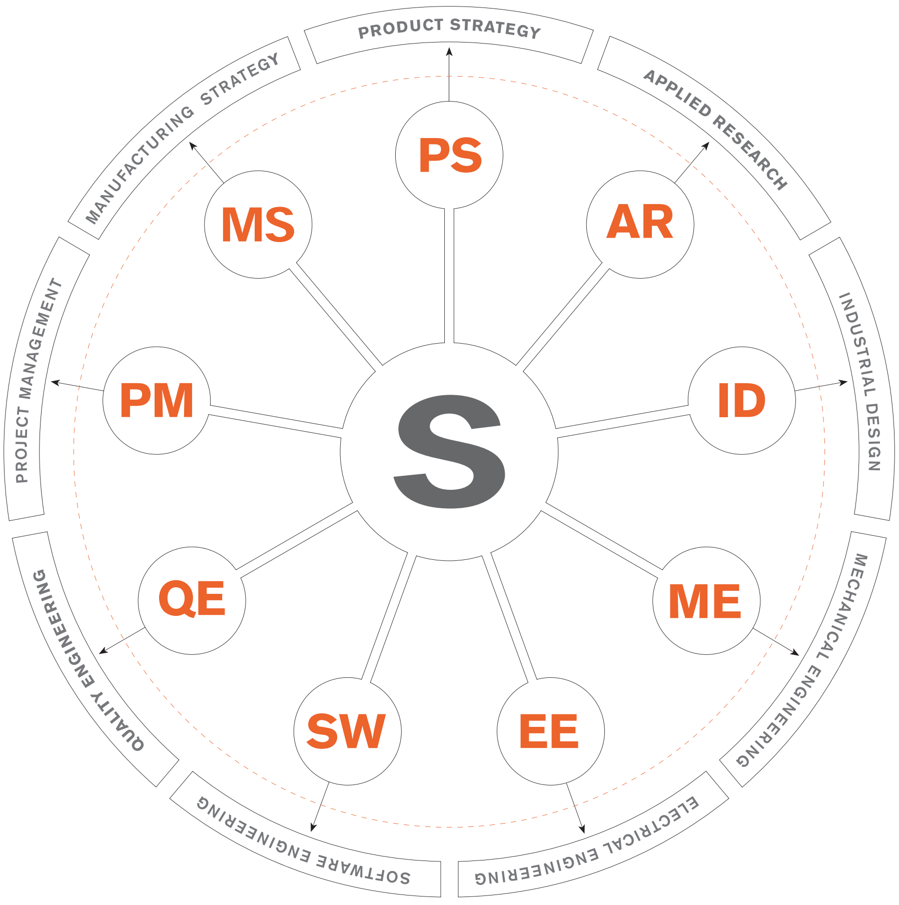
TAKE THE METROBUS OR METRORAIL TO THE USA SCIENCE & ENGINEERING FESTIVAL

M metro

STRATOS

INNOVATIVE PRODUCT DEVELOPMENT, ON DEMAND

>> We've spent 25 years building a superior ecosystem for innovation. If you need a full systems engineering team or a specialized area of expertise, Stratos is here to help.



designwest
center of the engineering universe

March 26-29, 2012
McEnery Convention Center
San Jose, CA

VISIT US AT **BOOTH #2314** AT DESIGN WEST 2012 TO LEARN MORE AND ENTER TO WIN A **3RD GENERATION IPAD™.**