UBM Electronics

Learn today. Design tomorrow.

ESD

EMBEDDED SYSTEMS DESIGN

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

VOLUME 24, NUMBER 2

MARCH 2011

Code analysis: Use the right tools

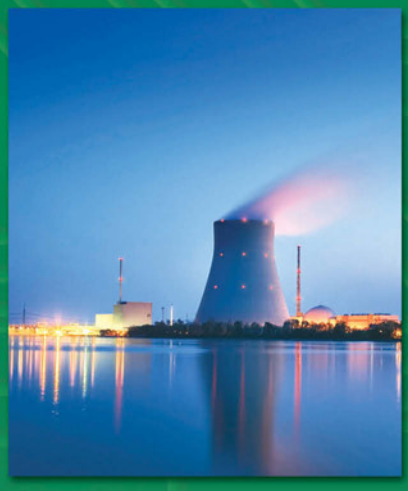
Leading Embedded Development Tools

The complete development environment for ARM®, Cortex™, 8051, and C166 microcontroller designs.

ARM® 1-800-348-8051 www.keil.com KEIL™ Tools by ARM

The INTEGRITY® RTOS

Certified and Deployed Technology



The INTEGRITY RTOS is deployed and certified to:

Railway: ***EN 50128 SWSIL 4***, certified: 2010

Security: ***EAL6+ High Robustness***, certified: 2008

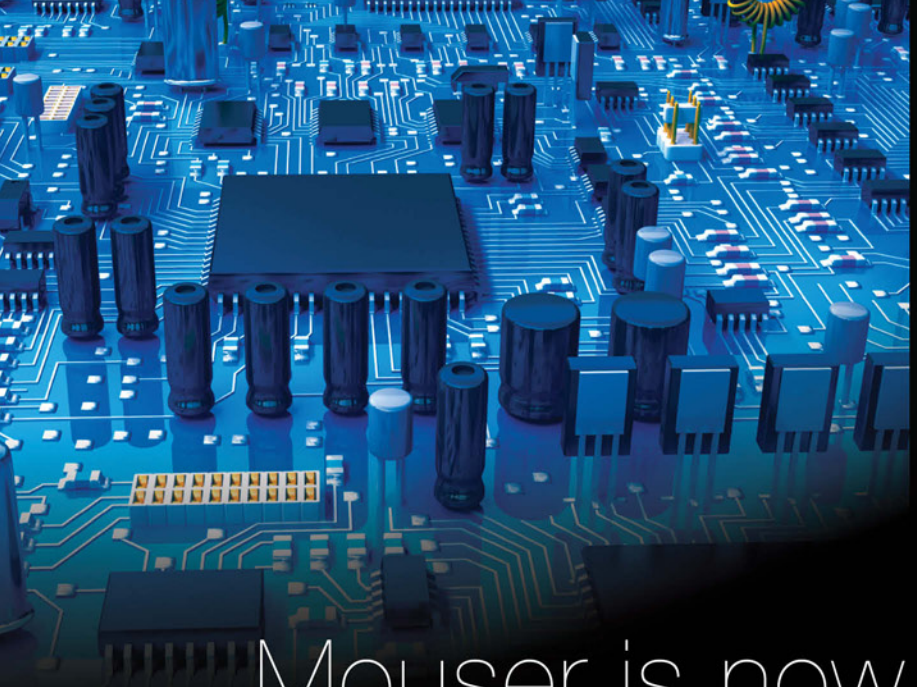
Medical: ***FDA Class III***, approved: 2007

Industrial: ***IEC 61508 SIL 3***, certified: 2006

Avionics: ***DO-178B Level A***, certified: 2002

Green Hills® SOFTWARE

www.ghs.com



Mouser is now an authorized distributor for **all** things Maxim.

Semiconductors,
Engineering
Development Tools,
Embedded Solutions,
Optoelectronics

Maxim Integrated Products and Mouser have teamed together to speed your development of breakthrough analog and mixed-signal engineering solutions. Discover a whole new range of possibilities. Get your hands on what's next from Mouser. To learn more, visit ***mouser.com/maxim***

Bipolar ADCs

PWM Step-Down
DC-DC Regulator

Authorized Distributor
of all Maxim products.

MAXIM
INNOVATION DELIVERED®

MOUSER ELECTRONICS
a tti company

The Newest Products for Your Newest Designs™

Learn today. Design tomorrow.

ESD

EMBEDDED SYSTEMS DESIGN

VOLUME 24, NUMBER 2
MARCH 2011

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, 600 Harrison Street, 5th floor, San Francisco, CA 94107, (415) 947-6000. Please direct advertising and editorial inquiries to this address. SUBSCRIPTION RATE for the United States is $55 for 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, EE Times Group.

COLUMNS

DEPARTMENTS

ESC Silicon Valley
May 2–5, 2011
San Jose, CA
http://esc-sv.techinsightsevents.com/

ESC Chicago
June 6–8, 2011
Chicago, IL
http://esc-chicago.techinsightsevents.com/

ESC India
July 20–22, 2011
Bangalore, India
www.esc-india.com/

ESC Boston
July 20–22, 2011
Boston, MA
http://esc-boston.techinsightsevents.com/

ONLINE

www.embedded.com

INDUSTRIAL

AEROSPACE

SYSTEM ON A CHIP

MEDICAL

AVIATION

CONSUMER

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

Express Logic has completed 14 years of successful business operation, and our flagship product, ThreadX, has been used in over 800 million 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.

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 new PrismX™ 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

ESD
EMBEDDED SYSTEMS DESIGN

Editorial Director
Ron Wilson
(415) 947-6317
ron.wilson@ubm.com

Managing Editor
Susan Rambo
susan.rambo@ubm.com

Acquisitions/Newsletter Editor, Embedded.com Site Editor
Bernard Cole
bccole@acm.org

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

Art Director
Debee Rommel
debee.rommel@ubm.com

Production Director
Donna Ambrosino
dambrosino@ubm-us.com

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

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

Associate Publisher/Sales North America
Bob Dumas
(516) 562-5742
bob.dumas@ubm.com

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

UBM Electronics

Corporate—EE Times Group

Paul Miller	*Chief Executive Officer*
Felicia Hamerman	*Group Marketing Director*
Brent Pearson	*Chief Information Officer*
Jean-Marie Enjuto	*Financial Director*
Amandeep Sandhu	*Manager Audience Engagement*
Barbara Couchois	*Vice President Sales Ops*

Corporate—UBM LLC

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

#include

BY Ron Wilson

Unintended acceleration

As we were preparing this month's issue for publication, the U.S. National Highway Traffic Safety Administration released an enormous report on its investigation into unintended acceleration in Toyota cars. It all seems very familiar, if you remember the Audi 5000. New electronic throttle control reaches the market. Reports of failure emerge from a few users. The vendor denies any problem. A few engineers quietly report having reproduced the problem. But intensive publicly-funded investigation finds nothing.

What makes this report particularly interesting is that the NHTSA called in an evaluation team from NASA to do the heavy lifting. And that team included a software evaluation group. While the hardware folks were shaking, baking, and irradiating cars and car parts, the software team had at the Engine Controller Module code for the four-cylinder 2005 Camry—all 280K lines of ANSI C. The team's report (*www.nhtsa.gov/UA*) could be a case study for Mark Pitchford's cover story.

NASA's team applied static source-code analysis, formal logic model checking, and algorithm analysis through simulation. The report states "The team's experience is that there is no single analysis technique today that can reliably intercept all vulnerabilities, but that it is strongly recommended to deploy a range of different leading tools."

For code analysis, the team used Coverity, CodeSonar, and Bell Labs' Uno to identify "common coding defects and suspicious coding patterns." They also used CodeSonar to compare Toyota's code against a Jet Propulsion Lab coding standard.

For model checking, the team used open-source Spin and Swarm. Here the tale gets more interesting. To use a formal model checker, you first have to write formal models. The team decided to build models only for those software modules they believed could be culprits—so the formal analysis depended upon human judgment of possible fault modes.

The algorithm analysis started with—once again—building models, this time in Matlab. This process started with reading Toyota documentation and talking with Toyota engineers, and then progressed to analyzing the source code and finally testing the models against actual Camrys. Once the NASA team was satisfied with the models, they explored failure scenarios in Simulink and checked delays with AbsInt aiT.

Some conclusions suggest themselves. First, there are no silver bullets: effective debug means using everything you've got. Second, even when it's grounded in exhaustive and formal techniques, an evaluation is circumscribed by the evaluators' beliefs about the possible behavior of the system. Third, there is no certainty. Despite Toyota's great care in developing their code, NASA's analysis found significant errors, including serious underestimates of delays in the multiprocessing system. But the investigation could not link those errors to any proposed mechanism for unintended acceleration. Contrary to what you probably read in the papers, the NASA Executive Summary stated "Because proof that the ETCS-i caused the reported UAs [unintended accelerations] was not found does not mean it could not occur."

—Ron Wilson, *ron.wilson@ubm.com*

Ron Wilson is the editorial director of design publications at UBM Electronics. You may reach him at ron.wilson@ubm.com.

Dreaming of push-button synthesis

You can look at the C-to-Si problem *(Ron Wilson, "C to silicon. Really?" January/February 2011, www.eetimes.com/4213003)* as being much the same as the "How do I get my C to run parallel problem?"—in other words, hardware descriptions are fine-grained parallel processing versions of an algorithm. Rather than RTL, you need to refine your software down to the level of small finite state machines communicating asynchronously, and then give that to your logic synthesis tool (for ASIC or FPGA).

http://parallel.cc

The push-button idea is not dead, but a lot of the work needs to be done by the software (compiler) guys rather than the EDA guys.

—DKC

Since SystemC has been around for some time and the dream that you describe is not yet real, is an intermediate step of any value? A lot has been said about the inherent parallelism in HDL, but is it really ever done to the max? In order to do things in parallel other than step counters, a lot of resource can be gobbled up by implementing arrays in registers and having adders all over the place, so a commonsense look may say that Utopia is not really worth the price.

Although it is generally accepted that pipelining is a must, a small number of CPU registers must be allocated by the compiler, and memory must be accessed one word at a time (maybe both an instruction and data together). These things lead to a lot of serialization in the CPU and then the need to carve out things that can be accelerated. But let's think about what is available on an FPGA. Internal multiport memory can be used to achieve parallel access to two operands and write the previous result in the same cycle. Existing CPUs do not use that capability, but I can show a way to take C code and do an iteration of a basic "for" loop in about 11 clock cycles at about the typical fmax of the FPGA. Yes, custom HDL will go faster, but not much because the cycle count is so small and the clock is running at the typical rate for the chip.

—KarlS

Push-button synthesis, even for something as mature as RTL synthesis, is not a realistic expectation if you are targeting production silicon. This is why Cadence focused much of the "C-to-Silicon Compiler" development on a graphical design environment where hardware engineers (yes, they are still needed!) can examine the Control/Dataflow Graph, the critical path from embedded RTL Compiler annotated to the SystemC, etc., and make the desired adjustments. Engineers need to optimize the synthesis of the SystemC for the end-application. The TI OMAP example points out that when they create the platform and subsystems, they aren't sure how they will be used in the future. If the source of a block/subsystem is SystemC, you can generate RTL's that are targeted for different product goals. But you still need engineers to drive high-level synthesis correctly, just as you still need engineers to drive RTL synthesis, floorplanning, and place and route even though so much automation has been added.

A commonsense look may say that Utopia is not really worth the price.

—Jack.Erickson

Coremark and multicore

One interesting thing about the CoreMark benchmark *(Shay Gal-On and Markus Levy, "CoreMark: A realistic way to benchmark CPU performance," January/February 2011, www.eetimes.com/4212735)* is that its performance is almost linearly scalable with the number of cores in a multicore processor. So the TilePro 64-core processor comes in at the top of the list with a CoreMark/MHz rating of more than three times its closest competitor. Of course, many (most?) real-world applications don't scale linearly with core count because of shared resources such as memory and I/O.

—HaroldR

Demystifying constructors

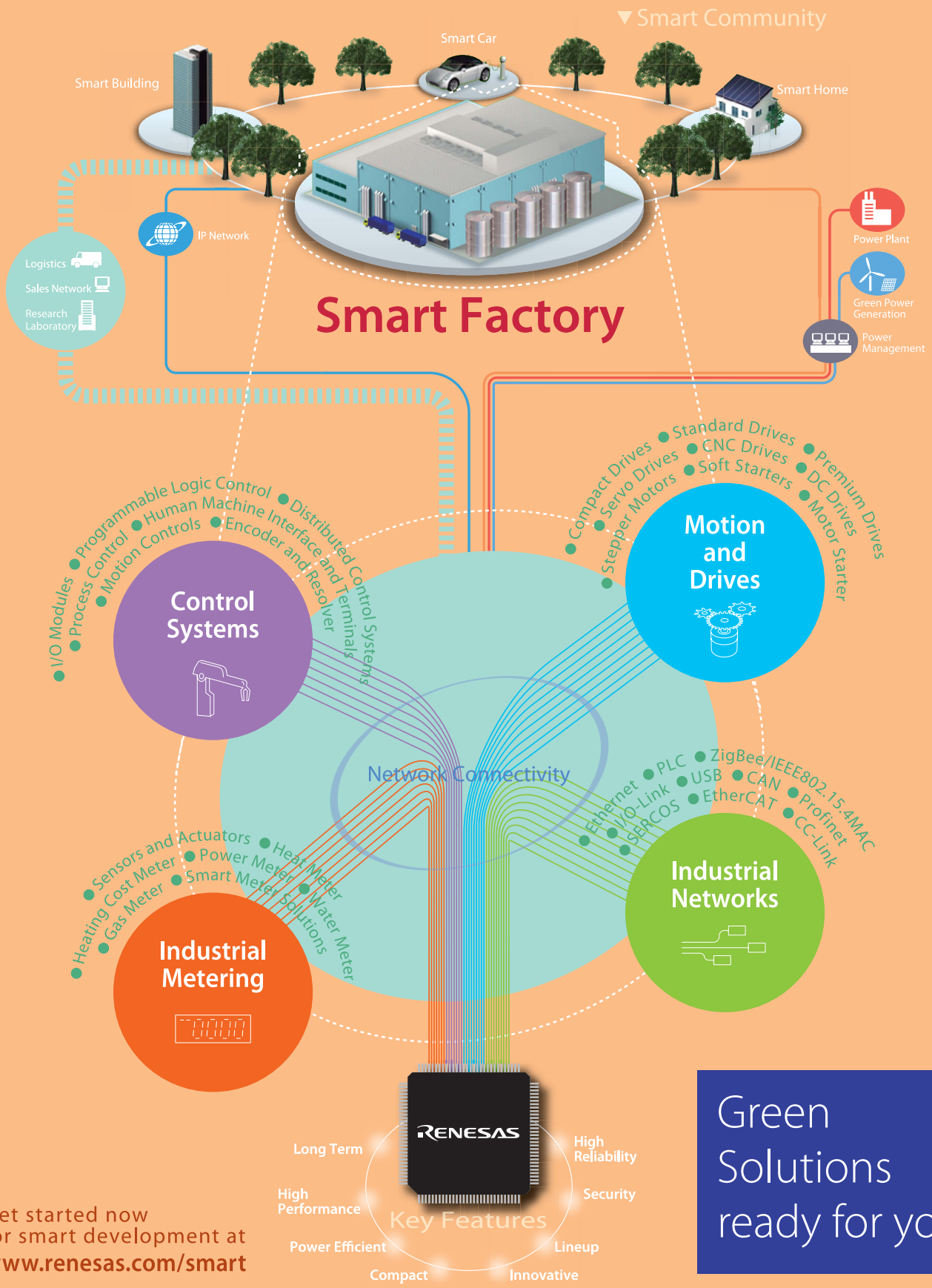
We found an effective way to get around the problem of constructors being called automatically when you did not expect *(Dan Saks, "Demystifying constructors," January/February 2011, www.eetimes.com/4212701)*. Make them private. That disables new. and prevents some terrible casts from happening. The class then provides a function to make an object (or objects)

```
x = class.newobj(...)
```

All objects are then explicitly created making the construction sequences much more visible.

—cdhmanning

We welcome your feedback. Letters to the editor may be edited. Send your comments to Ron Wilson at ron.wilson@ubm.com or post a comment online, under the article you wish to discuss. We edit letters and posts for brevity.



Green Solutions ready for you

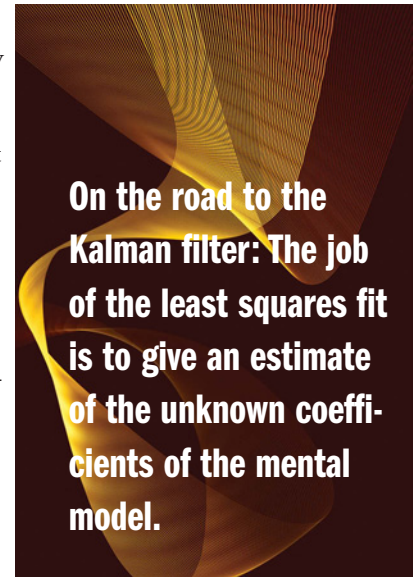
Get started now for smart development at
www.renesas.com/smart

Renesas Electronics Corporation

RENESAS

By Jack W. Crenshaw

Can you give me an estimate?



About 18 months ago, I wrote a column on the least squares fit ("Why all the math," *www.eetimes.com/4027019)* where I emphasized the technique to fit a curve to noisy data. It's not the only way to do that, of course. If you've ever worked with any embedded system that had a sensor measuring a real-world parameter, you know all about dealing with noise. The usual way is to pass the signal through a low-pass filter, which can be anything from a simple first-order lag to a finite impulse response (FIR) or infinite-impulse response (IIR) filter with scores, if not hundreds, of terms.

So what's the difference between a least squares fit and a low-pass filter? They both extract a signal from the noise, right?

One clue to the difference lies in the word, *signal*. As soon as you say the word, you invoke an image of something like a voltage that's changing with time. When you're processing a signal, you don't have the luxury of waiting until you've got a bunch of data points. You have to accept and process the data as it's coming in. Most days, we call such processing *real time*.

By contrast, the term *least squares* invokes an image of *batch processing*, where you operate on a whole set of data items after they've been collected. The notion of time is not explicitly involved. Indeed, when you're applying the least squares fit, you can shuffle the data indiscriminately, and the technique will still give you the same solution.

As far as we know, the first application of the least squares fit was invented by Carl Friedrich Gauss in 1795, at the ripe old age of 18. He used it in 1801 to predict the motion of the minor planet, Ceres. I think it's safe to say that Gauss's analysis was anything *but* real time. Unless, of course, you accept a clock rate measured in days or weeks.

The distinction between a filter and a least squares fit gets a lot more fuzzy if we set up the least squares fit so that we can process the data *sequentially*, as it comes in. That concept, sequential processing, is going to be our main focus in this and future columns.

But there's another distinction between filtering and fitting that's much more fundamental and profound than the way you process the data. That distinction lies in what you know—or think you know—about the process that generates the data.

When I'm developing a system to filter noise out of a signal, I don't have any idea what's happening in the real world, to create that signal. At the level of the analog-to-digital (A/D) conversion, it's just a voltage that comes from somewhere, corrupted by noise. My job is only to extract the signal from the noise.

The least squares fit is different. We apply it when we think we know something about the process that generated the data. If I'm applying a linear regression to a data set, it's because I think that one element of the set's data pair depends on the other. And not just depends, but has a *linear* relationship, which I can graph as a straight line. The purpose of the least squares fit is not just to filter the noise, but to discover the coefficients of that straight line relationship.

Of course, the regression doesn't have to be linear. Using least squares, I can fit a quadratic, or a polynomial of any order. I can fit a logarithmic relationship, or a sum of logarithmic terms. I can even fit a sum of sine waves (in which case I've done a Fourier analysis). It really doesn't matter what we think the relationship is; it only matters that we think that there is one.

The point is, when I'm applying a least squares fit I'm doing more than just filtering noise. I have a mental *model* of what's going on in the system that's generating the data. Presumably, that model includes coefficients whose values are unknown. The job of the least squares fit is to give me a best *estimate* of those coefficients. For obvious reasons, this process is often called *state estimation*, and it's this discipline that will occupy our interest in the rest of this column, and several more. If things work out right, the series will culminate in that dream of all estimators, the justly famous *Kalman Filter.* I'll leave it to you to ponder why the pinnacle technique of state estimation is called, not an estimator, but a filter.

Jack Crenshaw is a systems engineer and the author of Math Toolkit for Real-Time Programming. He holds a PhD in physics from Auburn University. E-mail him at jcrens@earthlink.net.

YOUR SCORE WAS MERELY AVERAGE

In this new series, I expect to cover a lot of ground, with math that's going to get heavier and heavier as we go along. The last thing I want to do is to leave some of you behind in the starting blocks. So in the spirit of No Reader Left Behind, I'm going to begin with the most ridiculously simple example I can think of. If things go as planned, I'll continue to escalate in such small steps that, like the proverbial frog simmering in the proverbial stew pot, you'll be cooking along with state estimation without really noticing how you got there. I'll start by defining a set of 10 numbers:

$$\mathbf{y} = [4, 2, 5, 3, 2, 4, 2, 6, 3, 4] \quad (1)$$

Your challenge is to compute their average value. Hey, you know how to do this. First, add all the numbers:

$$sum = 4+2+5+3+2+4+2+6+3+4 = 35 \quad (2)$$

Then divide by the number of terms:

$$\bar{\mathbf{y}} = \frac{sum}{10} = 3.5 \quad (3)$$

There. That wasn't so hard, was it?

Note the bar over the **y**, which is the symbol usually used to indicate an average value. It's not the only symbol for the average. Another one is $\langle y \rangle$, which is a little harder to miss. I actually prefer $\langle y \rangle$, precisely *because* it's harder to miss. But we'll stick with the "bar" notation here. Note also that, while **y** is a set of values, $\bar{y}$ is only a single scalar number.

We can generalize the process with a little math notation (stay with me, now):

$$\bar{y} = \frac{1}{10}\left(y_1 + y_2 + \ldots + y_{10}\right) \quad (4)$$

or, more generally, for a set of N values:

$$\bar{y} = \frac{1}{N}\left(y_1 + y_2 + \ldots + y_{N-1} + y_N\right) \quad (5)$$

Even better, the shorthand notation of the summation symbol:

$$\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i \quad (6)$$

Careful, now. I can see some of your eyes starting to glaze over, and your pupils shrinking down to pinpoints. This isn't rocket science (yet). Equation 6 says nothing at all that isn't said by Equation 5. It's just a shorter way of saying it. Because mathematicians and physicists are a lazy lot, we tend to go for the shorter way when given the chance. Stated in words, the summation sign of Equation 6 says "let the index i take on all the integer values from 1 through N. For each i, add the term y_i to the sum."

The problem with taking an average is that the computation is a batch process. To add up all those numbers, we have to have them all available.

If the summation sign makes you nervous, just remember that you can always expand it back out into the explicit sum of Equation 5. It just takes a little longer to write, that's all.

One last point: The average value of any set is often called its *mean*. Even more specifically, its *arithmetic mean* (because there are other kinds of means).

Why introduce another term? Can it be that we're so lazy we'd rather write four letters than seven? I guess it must be so. Deal with it.

SOME DO IT SEQUENTIALLY

The problem with taking an average using Equation 6 (or 5) is that the computation is basically a batch process. To add up all those numbers, we have to have them all available. In an embedded computer, this means that we must keep the potentially huge set of terms stored in memory.

Is there a better way? Of course. To see how, let's look at how the average value changes as new data comes in. Let m_n be the mean of the first n values of the set. As the data comes in, we'll have:

$$\begin{aligned} m_1 &= \frac{1}{1}y_1 = y_1 \\ m_2 &= \frac{1}{2}\left(y_1 + y_2\right) \\ m_3 &= \frac{1}{3}\left(y_1 + y_2 + y_3\right) \\ m_4 &= \frac{1}{4}\left(y_1 + y_2 + y_3 + y_4\right) \\ &etc. \end{aligned} \quad (7)$$

But each of the sums in these equations are merely the sum in the previous mean, plus one new term. We can write:

$$\begin{aligned} m_1 &= \frac{1}{1}y_1 = y_1 \\ m_2 &= \frac{1}{2}\left(1m_1 + y_2\right) \\ m_3 &= \frac{1}{3}\left(2m_2 + y_3\right) \\ m_4 &= \frac{1}{4}\left(3m_3 + y_4\right) \\ &etc. \end{aligned} \quad (8)$$

And, in general:

$$m_{n+1} = \frac{1}{n+1}\left(n \bullet m_n + y_{n+1}\right) \quad (9)$$

As you can see, we don't need to keep any of the old elements of **y** around. In fact, we don't need to keep any of the old elements of **m** around, either. We only need the latest value of m, plus the newest element of y. The new value of m can overwrite the old value. In a software implementation, we only need to have two persistent, scalar variables: the past value of m, plus the integer counter, k.

Better yet, let's not keep the old value of m, but the running sum. This lets us avoid the multiplication by n. Let:

$$s_n = \sum_{i=1}^{n} y_i \quad (10)$$

Then:

$$m_{n+1} = \frac{1}{n+1}\left(s_n + y_{n+1}\right) \quad (11)$$

Writing the software is almost easier than describing what it must do. **Listing 1** shows a snippet of code that works in either C or C++. It's not perfect, and it's not production quality. Because it has static variables, it won't work if you're asking it to find more than one average

Scan Here

mouser.com
Semiconductors and electronic components for design engineers.

The **most** advanced, multilingual, multicurrency mobile site for design engineers.

mousermobile™

Compatible with more than 25 mobile platforms, no one supports more phones and tablets than Mouser. Get What's Next now at **m.mouser.com**

mouser.com | The Newest Products For Your Newest Designs™

MOUSER ELECTRONICS®
a tti company

Listing 1 Compute the average.

```
/* Compute a running average of a set of input values
 *
 * Jack W. Crenshaw
 *
 * For illustration only.  Will only work if used for a single
 * data set.
 * For production, consider a C++ class
 */

double Avg(double x)
{
  static int n = 0;
  static double s = 0;
  s += x;
  n++;
  return s/n;
}
```

per program. If there were ever a case for a C++ class, this is it. That task, I'm leaving "as an exercise for the student." But the code I've shown should give you the idea.

WE NEED A VARIANCE

That task—of computing an average—wasn't too hard, was it? I hope it didn't tax your brains too much. Here's your next challenge: For the same data set, compute the variance and standard deviation.

Whoa! You want me to . . . what? Did we cover that in class?

Not yet, but we'll do it now. In general, the data in our sample data set **y** tells us more than just its mean, or average, value. It also tells us something about the reliability of each data element. In other words, it gives us insight into how much noise hides inside the data. It gives us the *statistics*.

In **Figure 1**, I've plotted the data set and also the mean value. As you can see, the data items themselves bounce around quite a bit. As often happens, none of them are actually equal to the mean value. For each value of **n**, the value in the set differs from the mean by an amount called the *residual*.

$$e_i = y_i - \bar{y} \tag{12}$$

It would be nice if we had some single scalar measure—we might even call it a *variance*—of the quality of the data. Clearly, this measure would have to involve all the members of the data set. We could try adding all the residuals together, or computing their average value, but that wouldn't work. Because the residuals can be positive or negative, they could cancel each other, leaving us with a false impression. If, for example, the data values alternated around the mean, then the sum of any pair—or all the pairs—would be zero, and so could be the average residual. That would tell us nothing about the real quality of the data set.

A measure that does work, though, is to take the sum of the *squares* of the residuals. This, of course, is the same measure that's used in linear regression and similar curve fits. Can we say "least *squares*"? Duh. More precisely, let's define *variance* as the mean of the squared residuals:

$$V = \frac{1}{N}\sum_{i=1}^{N} e_i^2 = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \bar{y}\right)^2 \tag{13}$$

Once we have the variance, the *standard deviation* is easy. It's just the square root of the variance:

$$\sigma = \sqrt{V} \tag{14}$$

This quantity, standard deviation, is supposed to be a measure of the amount of noise in our signal. Take a look at **Figure 2**. This is the same graph as Figure 1, but I've added the two horizontal, dashed lines a distance σ above and below the average value. You get the impression that new measurements of *y* are usually going to lie in the band between these two limits. Are the limits absolute? Certainly not. As you can see, five of the 10 points lie outside the band. Even so, these lines do seem to say something about the "scatter" in the data, don't they? From the defining equations, it's clear that the size of σ is going to depend on this scatter. If all the values y_i are nearly equal, then σ will be small, and the band will be more narrow. In the limit, when there

A plot of the data set and the mean value.

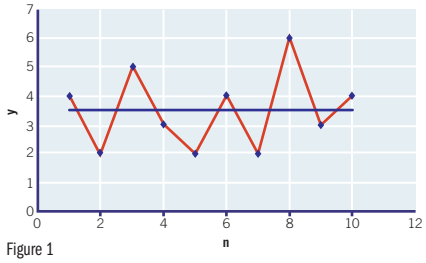


Figure 1

Example with error bounds.

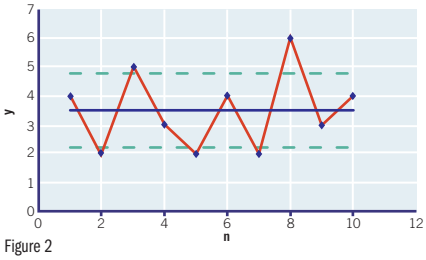


Figure 2

The sequential process.

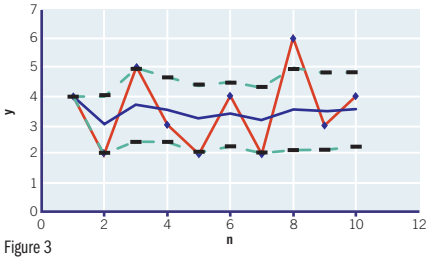


Figure 3

is no noise, or scatter, at all, all the measurements will be equal, the value of σ will go to zero, and so the band will have zero width.

You'll note that I haven't said anything, so far, about probabilities, or distribution functions, or any of those terms that relate to statistics. And I won't, in this column. We'll have plenty of time for that, later. For now, you only need to get the concept that the standard deviation is a measure of the amount of scatter, or noise, in the data.

A SEQUENTIAL APPROACH

We started this discussion by considering the batch processing of the average, assuming that we had the entire data set as embodied in **y**. If we look at the problem of computing the variance from the same perspective, it clearly isn't much harder. First, sum all the elements of **y,** and divide by *N* to get the mean $\bar{y}$. Then use that in Equation 12 to calculate the residuals term by term. Square them, add them, and divide by *N* to get the variance.

The question is, can we do the same thing for variance that we did for the mean? Can we come up with a sequential process for the variance?

At first glance, it may seem unlikely. The problem is that *all* the residuals, from e_1 through e_n, depend on the mean, which changes each time we add even one more data element. From Equation 13, it sure looks like we must recalculate the entire variance from scratch.

But looks can be deceiving. You can see the process with more clarity if we expand the polynomial in Equation 13. Write:

$$V = \frac{1}{N}\sum_{i=1}^{N}\left(y_i^2 - 2y_i\bar{y} + \bar{y}^2\right) \tag{15}$$

Now let's split up the three terms so that we sum them separately:

$$V = \sum_{i=1}^{N} y_i^2 - \sum_{i=1}^{N} 2y_i\bar{y} + \sum_{i=1}^{N}\bar{y}^2 \tag{16}$$

Listing 2 Compute the mean and variance.

```
/* Compute a running average and variance of a set of input
 * values
 *
 * Jack W. Crenshaw
 *
 * For illustration only.  Will only work if used for a single
 * data set.
 * For production, consider a C++ class
 */

void AvgVar(double x, double & mean, double & var)
{
     static int n = 0;
     static double sum = 0;
     static double sumSq = 0;

     sum += x;
     sumSq += x*x;
     n++;
      mean = sum/n;
     var = sumSq/n - mean*mean;
}
```

Because we know that $\bar{y}$ itself contains a summation, it's tempting to substitute that sum here. But that would be a mistake; it leads to more complexity, while we're looking for simplicity.

The real trick is to recognize that, regardless how we calculate $\bar{y}$, it's just a number by the time it shows up in Equation 16. It's not an indexed number, and it isn't changing as we generate the sums. As far as the summations are concerned, it's a mere constant, and we can boost it out of the summation process.

Doing that, we can write:

$$V = \frac{1}{N}\left(\sum_{i=1}^{N} y_i^2 - 2\bar{y}\sum_{i=1}^{N} y_i + \bar{y}^2\sum_{i=1}^{N} 1\right) \tag{17}$$

Check that last sum. Is that cool, or what? It simply says that we should add 1 to itself *N* times. Guess what result we'll get. That's right, it's *N*. The second sum is interesting, also. Does that sum:

$$\sum_{i=1}^{N} y_i$$

look familiar? It's the same sum that's used to generate the mean in the first place. See Equations 6 and 10. From them, we can see that:

$$\sum_{i=1}^{N} y_i = s_N = N\bar{y} \tag{18}$$

Substitute this into Equation 17 to get:

$$V = \frac{1}{N}\left[\sum_{i=1}^{N} y_i^2 - 2\bar{y}\left(N\bar{y}\right) + N\bar{y}^2\right]$$

Or:

$$V = \frac{1}{N}\sum_{i=1}^{N} y_i^2 - 2\bar{y}^2 + \bar{y}^2 \tag{19}$$

Or simply:

$$V = \frac{1}{N}\sum_{i=1}^{N} y_i^2 - \bar{y}^2 \tag{20}$$

We're now in a position to calculate the variance, and therefore the standard deviation, sequentially. We'll have to maintain another sum; call it:

$$u_n = \sum_{i=1}^{N} y_i^2 \tag{21}$$

Each time we get a new value of y_i, we can now compute the running sum s_n as before, plus the running sum of squares, u_n. From s_n, compute the latest value of $\bar{y}$. Plug that into Equation 20 to

CONTINUED ON PAGE 32

Static code analysis has been around as long as software itself, but you'd swear from current tradeshows that it was just invented. Here's how to choose the right code-analysis tools for your project.

Think static analysis cures all ills? Think again.

BY MARK PITCHFORD

Static analysis (or static code analysis) is a field full of contradictions and misconceptions. It's been around as long as software itself, but you'd swear from current trade shows that it was just invented. Static analysis checks the syntactic quality of high-level source code, and yet, as you can tell from listening to the recent buzz, its findings can be used to predict dynamic behavior. It is a precision tool in some contexts and yet in others, it harbors approximations.

With this extent of contradiction, it's hard to believe that all of these statements are accurate. *Static analysis*, a generic term, only indicates that the analysis of the software is performed without executing the code. So, simple peer review of source code fits the definition just as surely as the latest tools with their white papers full of various incantations of technobabble.

There isn't much point in any such analysis existing in isolation since even if code is perfectly written, it's only correct if it meets project requirements. It's therefore also important to understand how well any such analysis fits within the development lifecycle.

No analysis is good or bad simply by virtue of being static or dynamic in nature. It follows that each analysis tool is

```
0..2: Type of function to perform to calculate a value.
3..6: Type of data for the operation.
7     : Indicator for pre-computed or cached value.
```

```
class ring_buffer
    {
    ring_buffer();
    ring_buffer(size_t n);
    ~~~
    };
```

```
class ring_buffer
    {
    ring_buffer();
    ring_buffer(size_t n);
    ~~~
    };
```

```
procedure Linear_Search
   (Table : in     IntArray;
    Value : in     Integer;
    Found :    out Boolean;
    Index :    out Integer);
-# global in out Counter;
-# derives Counter, Found, Index from Counter, Table, Value
-# pre Counter < Integer'Last;
-# post Found -> (Table(Index) = Value and
-#                Counter = Counter~ + 1);
```

```
void rb_construct(ring_buffer *this, size_t n)
    {
    if ((this->base = (char *)malloc(n)) == NULL)
        /* return or announce failure more overtly */
    this->size = n;
    this->head = this->tail = 0;
    }
```

```
In Java:

Proc[] AllocProcessors (Config conf) {
  return new Proc[conf.num_proc];
}
```

```
type Proc_Number is range 1 .. 64;
type Proc_Array is array (Proc_Number range <>) of Proc;
type Proc_Array_Ptr is not null access Proc_Array;

procedure Alloc_Processors (Conf      : in Config;
                            Arr_Proc : out Proc_Array_Ptr) is
begin
   Arr_Proc := new Proc_Array (1 .. Conf.Num_Proc);
end Alloc_Processors;
```

neither good nor bad or perhaps more pertinently, appropriate or inappropriate just because they're statically or dynamically based. It's, then, important to look past the subtle advertising and self-congratulatory white paper proclamations to consider the relevant merits and demerits of static analysis and its ability to predict dynamic behavior. Can a solid static-analysis engine bypass the need for dynamic analysis? In this article, I explore current technologies and explain how static analysis predicts dynamic behavior. This article will help developers understand which method to use under which circumstances.

We'll look specifically at five key attributes of analysis tools, shown in the sidebars.

Test-tool vendors offer a plethora of combinations. Despite their lofty claims, no single vendor touts an offering that embraces all of these attributes.

The first three are attributes of static-analysis tools. Notably, these attributes don't comprehensively describe the categories of static-analysis tools, and many tools include more than one of these attributes.

The issue of what's static and dynamic analysis is further confused when there is a requirement to predict dynamic behavior. At that point, the dynamic analysis of code that has been compiled, linked, and executed offers an alternative to the prediction of dynamic behavior through static analysis.

Dynamic-analysis tools involve the compilation and execution of the source code either in its entirety or on a piecemeal basis. Again, while many different approaches can be included, these characteristics complete the list of the five key attributes that form the fundamental "toolbox of techniques."

Test-tool vendors offer a plethora of combinations of these key static and dynamic attributes and claim their particular combination to be invaluable to the efficient and effective development of software.

Despite their lofty claims, no single vendor touts an offering that embraces all of these attributes. And, attempting to apply every one of the five techniques through a combination of tools would usually be prohibitively expensive both in terms of capital investment for the tools and labor costs for software testing.

KEY ATTRIBUTES OF STATIC-ANALYSIS TOOLS

1. ***Automated code review***—automates the peer review process to enforce coding rules dictating coding style and naming conventions and to restrict commands available for developers to a safe subset.

   Code review doesn't predict dynamic behavior, except to the extent that code written in accordance with coding standards can be expected to include fewer flaws that might lead to dynamic failure.

2. ***Formally-defined language***—(such as SPARK Ada) defines desired component behavior and individual run-time requirements. This may be in the form of specially formatted comments in the native language that are ignored by a standard compiler but can be statically analyzed to show that the program is "well-formed," consistent with the design information included in its annotations, and has certain properties specified in those annotations. The annotations therefore make it possible to precisely predict dynamic behavior via static analysis.

   The Larch/C++ approach is similar in concept and uses a predicate-oriented interface language.

3. ***Prediction of dynamic behavior through static analysis***—models the high-level code to predict the probable behavior of the executable that would be generated from it. This approach builds an approximate mathematical model of the code and then simulates all possible execution paths through that model, mapping the flow of logic on those paths coupled with how and where data objects are created, used, and destroyed.

   This approximation is used to predict anomalous dynamic behavior that could possibly result in vulnerabilities, execution failure, or data corruption at run time.

CONSIDERING THE ALTERNATIVES

Given that none of the vendors are keen to highlight where their own offering falls short, some insight into how to reach such a decision on your own would surely be useful.

By considering which attributes are most appropriate for a particular situation, you then know which product to choose.

Although vendors make presentations assuming developers are to work on a virgin project where they can pick and choose what they like, that's often not the case. Many development projects enhance legacy code, interface to existing applications, are subject to the development methods of client organizations and their contractual obligations, or are restricted by time and budget.

The underlying direction of the organization for future projects also influences choices:

- Is this a quick fix for a problem project in the field? Is the search for a software-test tool that will resolve a mystery and occasional run-time error crash in final test?
- Maybe there is a development on the order books that involves legacy code requiring a one-off change for an age-old client, but which is unlikely to be used beyond that.
- Perhaps you have existing legacy code and want to raise the quality of software development on an ongoing basis for new developments and/or the existing code base.

Or perhaps there is a new project to consider, but the lessons learned from past problems suggest that ongoing enhancement of the software development process would be beneficial.

To address your particular situation, it's initially useful to consider how each of the five key attributes fits into the development process.

The diagram in **Figure 1** superimposes the different analysis techniques on a traditional "V" development model. Obviously, your particular project may use another development model. In truth, the analysis is model agnostic. A similar representation could be conceived for any other development process model—waterfall, iterative, agile, and so forth.

The extent to which it is desirable to cover all elements of the development cycle depends very much on the initial state of development and the desired outcome.

During the coding phase, the application of coding standards and hence the use of automated code review is the least controversial of the five attributes. Many tools help with this phase of development, so product selection hinges on:

- Support of specific standards you need to comply with (such as MISRA, IEC 62304, CERT C, or internal company or project standards). Note that where tools claim to cover the same standard, the number of rules checked for that standard will vary from one tool to another.
- How easily you can adopt the tool into your development process.

With new development the analysis can begin as soon as any code is written—no need to wait for a compilable code set, let alone a complete system.

HOW THE FIVE ARE USED

I describe tools slightly out of the order from which they appear in the sidebars:

Automated code review can be applied whether the code under development is for a new project, an enhancement, or a new application using existing code. With legacy applications, automated code review is particularly strong for presenting the logic and layout of such code in order to establish an understanding of how it works with a view to further development. On the other hand, with new development the analysis can begin as soon as any code is written—no need to wait for a compilable code set, let alone a complete system.

Formally-defined languages (or formal methods) are labor intensive and, although they have their benefits, tend to be limited to highly safety-critical applications where functional integrity is absolutely paramount over any financial consideration (such as flight-control systems). Even then, the alternatives outlined here often prove to be more financially prudent and offer similar levels of quality.

Unlike the prediction of dynamic behavior through static analysis, the use of Design by Contract principles often in the form of specially-formatted comments in the high-level code can accurately formalize and validate expected run-time behavior of source code.

Such an approach requires a formal and structured development process, textbook style, and uncompromising precision. What's more, applying such an approach to legacy code would involve a complete rewrite of it.

Unit testing and execution tracing focus on the behavior of an executing application and are therefore aspects of dynamic analysis. Unit, integration, and system analyses use code compiled and executed in a similar environment to that which is being used by the application under development.

Unit testing traditionally employs a bottom-up testing strategy in which units are tested and then integrated with other test units. In the course of such testing, individual test paths can be examined (*execution tracing*) to establish the most comprehensive coverage analysis. Clearly it's not necessary

KEY ATTRIBUTES OF DYNAMIC-ANALYSIS TOOLS

4. ***Execution tracing (or code coverage analysis)***—details which parts of compiled and linked code have been executed often by means of software instrumentation probes that are automatically added to the high-level source code before compilation.

5. ***Unit testing***—snippets of software code are compiled, linked, and built in order that test data (also called "vectors") can be specified and checked against expectations.

   Unit testing can be extended to include the automatic definition of test vectors by the unit test tool itself.

to have a complete code set available in order to initiate tests such as these.

Unit testing is complemented by functional testing, a form of top-down testing. *Functional testing* executes functional test cases, perhaps in a simulator or in a target environment, at system or subsystem level.

Clearly, these dynamic approaches test not only the source code, but also the compiler, linker, development environment, and potentially even target hardware. When the functionality of the code is the primary concern, you have little alternative but to deploy dynamic analysis. Unit test or system test must deploy dynamic analysis to prove that the software actually does what it is meant to do.

Alternatives do exist when robustness testing is the key concern, however.

The **static prediction of dynamic behavior** works well for existing code or less rigorously-developed applications. It doesn't rely on a formal development approach and can simply be applied to the source code as it stands, even when you have no indepth knowledge of the code. That ability makes this methodology very appealing for a development team in a fix—perhaps when timescales are short, but catastrophic and unpredictable runtime errors keep coming up during system test.

! The goal of finding every defect in a nontrivial program is unreachable unless approximations are included, which lead to "false positive" warnings.

Prediction of dynamic behavior via static analysis can take a number of forms, but it commonly uses the software source code as a model to predict the behavior of that code when it's executed. Each operation is statically evaluated against a superset of the whole range of operating conditions that can occur during program execution. The developer can then analyze the whole data set applicable to the

The five fundamental test-tool attributes directly relate to the specific development stages of design, code, test, and verification.

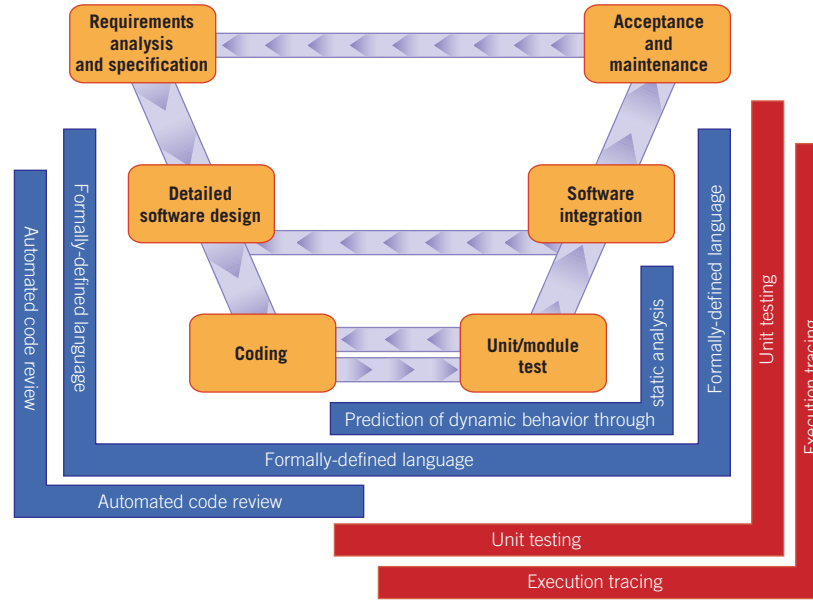


Figure 1

OUR PEOPLE
set us apart

and connect you with the right part, at the right price, at the right time.

Our vast line card features components and solutions from the world's leading manufacturers, and our renowned services support you through our distinguished technical expertise and spot-on execution.

Arrow Electronics provides the innovative technologies, services, and solutions you need to get to market fast.

Visit www.arrownac.com to:

- Search our more than 4.5 million parts and purchase with ease
- Hear about the latest new products
- Sign up for on-demand or live technical seminars and events
- Learn about market-specific solutions
- Take advantage of our engineering expertise and services
- Sign up for Arrow's technical enewsletters

arrownac

myarrow™
www.arrownac.com/myarrow

NEDA Supplier Authorized Distributor

code under test, rather than discrete data points. Because this approach works with a model, developers can potentially use this technique before any code can be built and run. Such an advantage makes it appear to offer a universal solution.

There is, however, a major downside. The code itself is not executing, but instead is being used as the basis for a mathematical model. As proven by the works of Church, Gödel, and Turing in the 1930s, it is an unfortunate fact that the resulting mathematical model is *always* an approximation—and what's more, even with future enhancements or entirely new developments that will *always* be the case for the static prediction of dynamic behavior.

It will never be possible for the model to be a precise representation of the code because such representation is mathematical insoluble for all but the most trivial examples. In other words, the goal of finding every defect in a nontrivial program is unreachable unless approximations are included, which by definition will lead to "false positive" warnings.

The complexity of the mathematical model also increases disproportionately to the size of the code sample under analysis. This is often addressed by the application of simpler mathematical modeling for larger code samples in order to keep the processing time for the analysis within reasonable bounds. But the simplifications can increase the number of these "false positives," which has a significant impact on the time required to interpret results. This trade-off can make the whole static-prediction approach unusable for complex applications.

! Dynamic unit and system testing enable you to prove that the code is robust and does what it should do in the environment where it will operate.

DYNAMIC APPROACH

Another alternative to the static prediction of dynamic behavior involves the automatic definition of test vectors using unit-test tools and hence performing dynamic analysis.

Unlike the static prediction of dynamic behavior, this dynamic approach does not and can never analyze the whole data set applicable to the

MAINTAIN THE BIDIRECTIONAL TRACEABILITY OF REQUIREMENTS

The intent of this specific practice is to maintain the bidirectional traceability of requirements for each level of product decomposition.

When the requirements are managed well, traceability can be established from the source requirement to its lower-level requirements and from the lower-level requirements back to their source. Such bidirectional traceability helps determine that all source requirements have been completely addressed and that all lower level requirements can be traced to a valid source.

Requirements traceability can also cover the relationships to other entities such as intermediate and final work products, changes in design documentation, and test plans. (ISO 26262 standard)

code under test. It does, however, involve the intelligent analysis of likely problem values such as boundary and inflection conditions, and specifically includes both them and other target values as it automatically generates the test vectors.

It's significant that this technique is deploying most (or potentially all) of the development environment and hence testing something that reflects the finished product much more accurately than the static prediction of dynamic behavior. Despite that advantage, it too can be deployed very early in the development cycle.

WHAT ARE WE TESTING HERE?

Perhaps the most telling point with regards to the testing of dynamic behavior—whether by static or dynamic analysis—is precisely what is being tested. Intuitively, a mathematical model with inherent approximations suggests far more room for uncertainty compared with code being compiled and executed in its native target environment.

If the requirement is for a quick-fix solution for some legacy code that will find most problems without involving a deep understanding of the code, the prediction of dynamic behavior via static analysis has merit. Similarly, this approach offers quick results for completed code that is subject to occasional dynamic failure in the field.

However, if you need to prove not only the functionality and robustness of the code but also provide a logical and coherent development environment along with an integrated and progressive development process, it makes more sense to use dynamic unit and system testing. *Dynamic unit and system testing* enable you to prove that the code is robust and does what it should do in the environment where it will ultimately operate.

As soon as the process becomes a critical factor, the case for an extensive dynamic element to test is compelling.

REQUIREMENTS MANAGEMENT AND TRACEABILITY

Most test tools ignore the requirements element of software development. That is reflected in **Figure 1**, in that none of the five key attributes directly covers requirement traceability at all. But the fact is that static and dynamic analyses fail to prove compliance with the functional requirements of the system. Even the best static and dynamic analyses will not prove that the software fulfills its requirements.

Widely accepted as a development best practice, *requirements traceability* ensures that all requirements are implemented and that all development artifacts can be traced back to one or more requirements. Most static and dynamic analysis vendors fail to provide what is needed by modern standards such as the automotive industry's draft standard ISO/DIS 26262 or the medical industry's IEC 62304 requiring bidirectional traceability. These standards place constant emphasis on the need for the derivation of one development tier from the one above it.

Such an approach lends itself to the model of a continuous and progressive use first of automated code review, followed by unit test and subsequently system test with its execution tracing capability to ensure that all code functions just as the requirements dictate, even on the target hardware itself—a requirement for the more stringent levels of most such standard.

While this is and always has been a laudable principle, last minute changes of requirements or code made to correct problems identified during test tend to put such ideals in disarray.

Despite good intentions, many projects fall into a pattern of disjointed software development in which requirements, design, implementation, and testing artifacts are produced from

isolated development phases. Such isolation results in tenuous links among requirements, the development stages, and the development teams.

The traditional view of software development shows each phase flowing into the next, perhaps with feedback to earlier phases, and a surrounding framework of configuration management and process (such as Agile and the Rational Unified Process). Traceability is assumed to be part of the relationships between phases. However, the reality is that while each individual phase may be conducted efficiently, the links between development tiers become increasingly poorly maintained over the duration of projects.

The answer to this conundrum lies in the requirements traceability matrix (RTM), shown in **Figure 2**, which sits at the heart of any project even if it's not identified as such. Whether the links are physically recorded and managed, they still exist. For example, a developer creates a link simply by reading a design specification and using that to drive the implementation.

This alternative view of the development landscape illustrates the importance that should be attached to the RTM. Due to this fundamental centrality, it's vital that project managers place sufficient priority on investing in tooling for RTM construction. The RTM must also be represented explicitly in any lifecycle model to emphasize its importance as **Figure 3** illustrates. With this elevated focus, the RTM is constructed and maintained efficiently and accurately.

When the RTM becomes the center of the development process, it has an impact on all stages of design from high-level requirements through to target-based deployment.

RTM sits at the heart of the project defining and describing the interaction between the design, code, test, and verification stages of development.

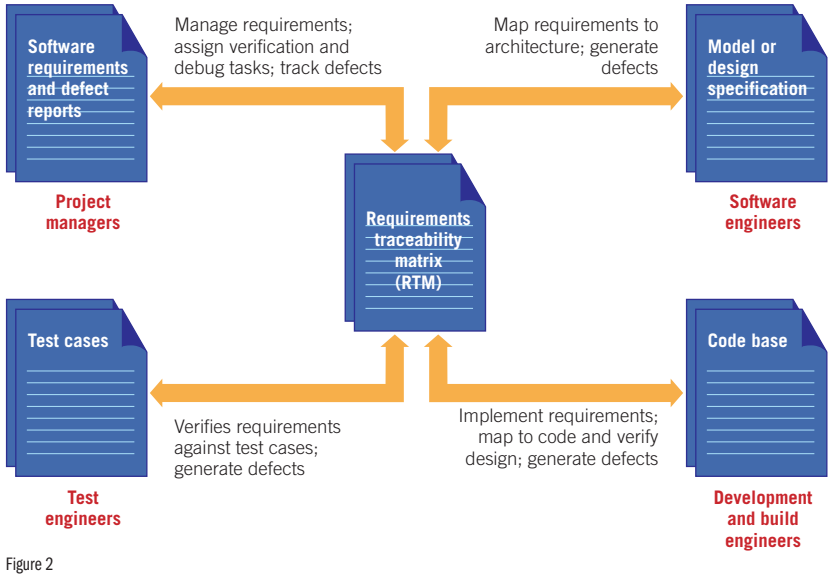


Figure 2

The requirements traceability matrix (RTM) plays a central role in a development lifecycle model. Artifacts at all stages of development are linked directly to requirements matrix and changes within each phase automatically update the RTM so that overall development progress is evident from design through coding and test.

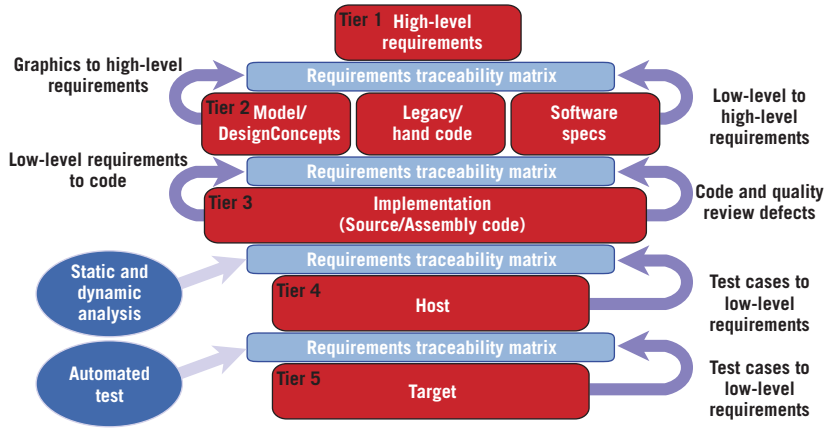


Figure 3

- The Tier 1 high-level requirements might consist of a definitive statement of the system to be developed. This tier may be subdivided depending on the scale and complexity of the system.
- Tier 2 describes the design of the system level defined by Tier 1. Above all, this level must establish links or traceability with Level 1 and begin the process of constructing the RTM. It involves the capture of low-level requirements that are specific to the design and implementation and have no impact on the functional criteria of the system.
- Tier 3's implementation refers to the source/assembly code devel-

You Dream it.

We Can Make it Wireless with ModFLEX™

ModFLEX™ gives you the
POWER & FREEDOM
to bring your wireless
product to market
YOUR WAY.

OPTION 1:
Purchase certified off the shelf modules.

OPTION 2:
License certified design files at anytime.

*LS Research offers royalty free license options.

The leader in wireless product development

ModFLEX™ by LS RESEARCH

Learn more at: www.lsr.com/products

oped in accordance with Tier 2. Verification activities include code rule checking and quality analysis. Maintenance of the RTM presents many challenges at this level as tracing requirements to source code files may not be specific enough and developers may need to link to individual functions.

In many cases, the system is likely to involve several functions. Traceability of those functions back to Tier 2 requirements includes many-to-few relationships. It's very easy to overlook one or more of these relationships in a manually managed matrix.

- In Tier 4 host-based verification, formal verification begins. Once code has been proven to meet the relevant coding standards using automated code review, unit, integration and system tests may be included in a test strategy that may be top-down, bottom up, or a combination of both. Software simulation techniques help create automated test harnesses and test case generators as necessary, and execution histories provide evidence of the "testedness" of the code.

Such testing could be supplemented with robustness testing if required, perhaps by means of the automatic definition of unit test vectors, or through the use of the static prediction of dynamic behavior.

Test cases from Tier 4 should be repeatable at Tier 5 if required.

At this stage, we confirm that the software is functioning as intended within its development environment, even though there is no guarantee it will work when in its target environment. However, testing in the host environment first allows the time-consuming target test to merely confirm that the tests remain sound in the target environment.

- Tier 5's target-based verification represents the on-target testing element of formal verification. This frequently consists of a simple confirmation that the host-based verification performed previously can be duplicated in the target environment, although some tests may only be applicable in that environment itself.

Where reliability is paramount and budgets permit, the static analysis of dynamic behavior with its "full range" data sets would undoubtedly provide a complementary tool for such an approach. However, dynamic analysis would remain key to the process.

Where reliability is paramount and budgets permit, the static analysis of dynamic behavior would provide a complementary tool

WHICH APPROACHES SHOULD YOU CHOOSE?

Each of the five key test tool attributes has merit.

There is a sound argument which supports traditional formal methods, but the development overheads for such an approach and the difficulty involved in applying it retrospectively to existing code limits its usefulness to the highly safety critical market.

Automated code review checks for the adherence to coding standards, and is likely to be useful in almost all development environments.

Of the remaining approaches, dynamic analysis techniques provide a test environment much more representative of the final application than static predictions of dynamic analysis and offers the means to provide functional testing.

Where requirements traceability is key within a managed and controlled development environment, the progressive nature of automated code review followed by unit, integration, and system test aligns well within the overall tiered concept of most modern standards. It also fulfills the frequent requirement or recommendation to exercise the code in its target environment.

Where robustness testing is considered desirable and justified, it can be provided by means of the automatic definition of unit-test vectors, or through the use of the static prediction of dynamic behavior. Each of these techniques has its own merits, with the former exercising code in its target environment, and the latter providing a means to exercise the full data set rather than discrete test vectors. Where budgetary constraints permit, these mutually exclusive benefits could justify the application of both techniques. Otherwise, the multifunctional nature of the unit-test tool makes it a cost effective approach.

If there is a secondary desire to evolve corporate processes towards the current best practise, both automated code review and dynamic analysis techniques have a key role to play in requirements management and traceability, with the latter being essential to show that the code meets its functional objectives.

If the aim is to find a pragmatic solution to cut down on the number of issues displayed by a problem application in the field, each of the robustness techniques—that is, the static analysis of dynamic behavior and the automatic definition of unit-test vectors—has the potential to isolate tricky problems in an efficient manner. ■

Mark Pitchford is a field applications engineer specializing in software test with with LDRA. He has over 25 years' experience in software development for engineering applications, the majority of which have involved the extension of existing code bases.

iStockphoto

Research shows that preemption-threshold scheduling helps to mitigate the deadline-vs.-overhead tradeoff faced by developers of real-time systems.

Lower the overhead in RTOS scheduling

BY ALEXANDER G. DEAN

Engineers creating real-time embedded applications typically use a real-time operating system (RTOS) to develop a system as a collection of independent tasks or threads, while maintaining responsiveness to time-critical events. These threads must run periodically or in response to some event and must complete before a certain deadline. Hard real-time systems demand that deadlines be met with absolute certainty, even under worst-case conditions. However, the need to meet deadlines can also result in increased overhead, conflicting with a real-time system's demand for low overhead. One technology—preemption-threshold scheduling (PTS)—reduces overhead while maintaining the ability to meet deadlines under worst-case conditions. In this article, I explain what PTS is, how it works, and how PTS addresses the real-time programming challenges facing embedded system designers.

Preemption-threshold scheduling reduces the number of context-switches by inhibiting preemption for a thread-specific range of priorities.[1] PTS preserves the ability to meet deadlines 100% of the time, with fewer high-overhead context switches. Developers can elect where and when to use PTS and otherwise allow traditional preemptive scheduling to occur. PTS gives them control over the responsiveness of each system thread and enables them to guarantee that all threads meet their individual deadlines.

PTS also offers other valuable benefits for resource-constrained systems. PTS is optimal among all preemption-limiting

ESC
Silicon Valley • May 2-5, 2011

Join the industry's leading embedded systems event
(Translation: Can't miss event)

ESC brings together the largest community of designers, technologists, business leaders and suppliers in one place.

Keynote Speaker

Steve Wozniak

**Co-Founder, Apple Computer, Inc.
and electronics industry visionary delivers the opening keynote speech on Tuesday, May 3rd at ESC Silicon Valley!**

Categories and Tracks that address the most relevant issues facing engineers and the industry. Take a moment to review this content, so that you can customize your educational experience.

Applications
- HMI and Multimedia
- Systems Architecture
- Reliability, Security, and Perforance
- Remote Monitoring and Wireless Networking

Embedded Software
- Linux/Android/Open Source
- Programming for Storage, I/O, and Networking
- Programming Languages and Techniques
- RTOS and Real Time Software
- Software Processes and Tools
- Windows for Embedded
- Quality Design and Untellectual Property
- Safety Design

Hardware for Embedded Systems
- Challenges and Solutions in Embedded Designs
- Connectivity and Security
- Memory in Embedded Designs
- Microcontrollers in Embedded Designs
- Powering Embedded Designs
- Programmable Logic in Embedded Designs

Tools and Best Practices
- Best Practices
- Debugging and Optimizing
- Design and Test
- Managing and Process
- Tools

Topics in Embedded-System Design
- Architecture Design
- DSP, Communication, and Control Design
- HW and Platform Design
- Quality Design and Intellectual Property
- Safety Design

Did you know that Steve Wozniak is an ESC Alumni?
"It will be good to reconnect with the engineering community that still drives so much electronics innovation."
Steve Wozniak
Co-Founder, Apple Computer, Inc. and Chief Scientist for Fusion-IO

Register today and save 10% OFF any package when you enter the promo code: sw
www.embedded.com/cc

Learn today. Design tomorrow.

ESC
Silicon Valley • May 2-5, 2011

UBM Electronics

methods for minimizing a system's total stack memory requirements—critical for systems with many threads. PTS also provides a useful mechanism for avoiding priority inversion.

WHY SMARTER PREEMPTION?

Most RTOSes use fully-preemptive schedulers where a task can preempt any lower-priority task at any time. Preemption allows critical processing to occur sooner than less-critical processing, delaying other processing until time is available. An interrupt service routine (ISR) is an example of such time-critical processing, but the same concept is typically extended to cover all tasks in the system. Preemption reduces the response time of critical processing in the system. A shorter response time can be used to achieve a more responsive system or more sophisticated processing. Alternatively, the processor clock speed can be reduced to save power, energy, or cost (through the use of a less expensive processor).

However, preemption has drawbacks:

- Each preemption incurs processing overhead. The scheduler must decide which task to run, and then swap processor states. Modern embedded processors typically have some hardware support to reduce this delay, but it isn't eliminated completely.
- Since preemptions can occur asynchronously among tasks, the system needs enough RAM to fit all worst-case stacks simultaneously. This can be prohibitively expensive for systems with little RAM or many tasks. This stack space must be large enough to accommodate worst-case function call nesting, local variable allocation, ISR activity, and possibly the task control block and context. Adding features such as a GUI, embedded web server, and TCP/IP stack will add multiple threads, each needing its own stack. Even in systems with larger amounts of RAM, running out of memory can be a common problem. Embedded systems usually evolve over time, with each new generation of software accreting a new layer onto the existing code base. This is also known as "The Software Gas Law." Hence, over a long enough time the data memory requirements can fill available memory, making RAM a precious resource. This makes the use of real-time kernels for low-end embedded systems particularly hard if not impossible.

Each preemption incurs processing overhead. The scheduler must decide which task to run, and then swap processor states.

- Systems with caches suffer from cache pollution by other tasks ("cache-related preemption delay") unless precautions are taken. Unfortunately, these precautions reduce performance. As the preempting Task B executes, it can evict data and instructions of the preempted Task A. When Task A resumes execution, it runs more slowly due to increased cache misses, which greatly complicates the worst-case timing analysis, since the preemption can typically occur at any place within Task A. Worst-case timing analysis is critical to ensuring real-time systems will meet their deadlines.
- Many embedded systems use dynamic voltage and frequency scaling to perform processing with as little energy or power as possible. Each task has its optimal frequency/voltage operating point and requires certain peripheral devices to be powered up. Allowing preemption introduces the possibility that the RTOS needs to transition to a new operating voltage and frequency, which further increases delays. Preempted tasks may have been using peripherals that need to remain powered during the preemption to maintain their state. Trying to guarantee the system will meet real-time deadlines even with delays to change the state of power management makes the analysis even more difficult.

To summarize, preemption improves system responsiveness at the cost of greater memory requirements and more difficult timing analysis and performance optimization.

PARTIAL PREEMPTION

Preemption need not be an all-or-nothing design decision. Many systems can in fact meet all deadlines without full preemption among tasks. In fact, there is a spectrum of possible approaches, ranging from nonpreemptable to fully preemptable. Researchers have been actively investigating this field and determining how to mathematically model the resulting systems in order to understand the design trade-offs better.

PTS is a promising technique for limiting preemptions.[2] PTS tries to minimize preemptions as much as possible while preserving the system's schedulability. (Real-time analysis techniques enable us to determine mathematically if a set of tasks is schedula-

Definitions of terms.

Term	Definition
DVS	Dynamic voltage scaling
DM	Deadline monotonic (scheduling)
EDF	Earliest deadline first (scheduling)
ISR	Interrupt service routine
MPTAA	Maximal preemption threshold assignment algorithm
PCP	Priority ceiling protocol
PIP	Priority inheritance protocol
PTS	Preemption-threshold scheduling
RM	Rate monotonic (scheduling)
SRP	Stack resource policy
SRPT	Stack resource policy with preemption-thresholds
WCET	Worst-case execution time
WCRT	Worst-case response time

Table 1

Find Your Next **Big** Idea


EE Times CONFIDENTIAL
Actionable intelligence for the global electronics industry
October 15, 2010

The Big Idea
Boardrooms in the spotlight
Lay of the Land
Next-gen battery battle page 4
IP Landscape
Trident, Zoran frame a debate page 7
EET on the QT
Aquantia circles wagons; IBM fab sale?; Commercial space boost page 9
VC Watch
VC universe shrinks page 12
Samplify stretches page 14
Market Data
Digital China shakes boxes page 16

Cable box market shifts east
Global set-top market shares
70%
60%
50%
40%
30%
20%
10%
0%
2010
2011
2012
Motorola
Pace
Cisco
Changhong
Thomson
Jiuzhou
Source: Digital Tech Consulting
Story, page 16

THE BIG IDEA

Accountability Comes to the Boardroom

The scandal involving former Hewlett-Packard CEO Mark Hurd has trained a spotlight on the role of corporate directorial boards and how they should function in a crisis.

Corporate boards seldom attract much attention, generally operating behind the scenes and, in theory, representing shareholders. Even before the Hurd scandal, which allegedly involved falsified expense reports and an improper relationship with a contractor, HP's board had been embroiled in a variety of high-profile controversies, including clashes with former CEO Carly Fiorina (now a U.S. Senate candidate in California) and a scandal involving spying on journalists that ultimately led to the resignation of then-chairwoman Patricia Dunn.

HP's latest fiasco illustrates what several

The HP fiasco illustrates what experts agree is a reliable rule of thumb: When a board makes the news, it's rarely good for the company

How much is that big idea worth?
Priceless, some might say. *EE Times Confidential* provides the insight you need to drive your business forward. With fast-paced analysis of the global electronics industry, *EE Times Confidential* tracks the latest trends and market research results, deciphers what it all means and provides actionable intelligence you can't find in a Google search.

Breakthrough— first.
Get a leg up on your competition and find that next big idea. You'll profit from unique insights and actionable intelligence you simply cannot find anywhere else.

Subscribe Now

www.eetimesconfidential.com

For more information contact Linda Uslaner
Linda.uslaner@ubm.com or 516.562.5843

***EE Times Confidential's* incisive analysis of the global electronics industry includes:**

- New business models, ecosystems and technology/product trends
- Quick but thorough reports on markets, technology and business issues
- Tracking VC investment, startup births and deaths; new industry trends
- The inside scoop on business deals and new technology partnerships. Find out which executive is moving where
- Analysis of what's behind the news
- Market intelligence, including pricing, inventory, production and market movements

EE Times
CONFIDENTIAL
Actionable intelligence for the global electronics industry

ble—all of its tasks will meet their deadlines in all possible scheduling situations—a fundamental requirement for many embedded systems.)

Researchers have been trying to understand how and when preemptions can be reduced or even eliminated. Baker introduced the Stack Resource Policy (SRP).[3] Gai et al. extended it to support PTS, resulting in the stack resource policy with preemption-thresholds (SRPT).[4] SRP and SRPT prevent priority inversions and deadlocks and bound the maximum time any task can be blocked.

PREEMPTION-THRESHOLD SCHEDULING

Normal fully-preemptive scheduling assigns a single priority to each task. This priority determines two aspects of the task's behavior—whether it can preempt another task and whether it can be preempted by another task. PTS assigns a separate priority to each aspect: The nominal priority determines whether this task can preempt other tasks, and the preemption-threshold sets this task's effective priority while executing.

When the task begins execution, its priority is raised to its preemption-threshold. In this way, all the tasks with priorities less than or equal to the preemption-threshold of the executing task cannot preempt it. PTS effectively creates groups of tasks that are not allowed to preempt each other.

It's easily seen that fully-preemptive and fully-nonpreemptive scheduling policies are special boundary cases of PTS. By assigning the preemption-threshold of each task equal to its priority, PTS simplifies to a fully-preemptive scheduling policy. By assigning the preemption-threshold of each task equal to the system's highest priority, PTS simplifies to a fully-nonpreemptive policy.

ASSIGNING PREEMPTION-THRESHOLDS

Preemption between tasks can be limited to occur only when necessary to maintain system schedulability. Tasks that run nonpreemptively with respect to each other can be mapped into the same run-time thread and share the same run-time stack, minimizing the memory requirements and other preemption overheads.

Preemption-threshold assignment works by starting with a default, fully-preemptive system—each task's preemption-threshold is equal to its preemption level. This is also called the *identity assignment*.

Starting with the lowest-priority task (setting i to N in Line 2 of **Listing 1**), each task's preemption-threshold γ_i is set to j and the system is evaluated for schedulability. This preemption-threshold j (and therefore γ_i) is incremented until any further increase would make the system unschedulable. Wang and Saksena developed a heuristic shown in Listing 1 that can always find a feasible preemption-threshold assignment if it exists with a time complexity of $O(N^2 \cdot q(N))$ where $q(N)$ is the complexity of the schedulability checking function. We call this algorithm the *maximal preemption-threshold assignment algorithm (MPTAA)* because it will always find the preemption-threshold assignment that is larger than any other feasible preemption-threshold assignment.

Listing 1 Modified maximal preemption-threshold assignment algorithm.

```
Algorithm 1 MPTAA(T , Π)
 1: Γ = Γ^I      /* initialize preemption threshold to identity
                    assignment */
 2: for i = N down to 1 do
 3:   j = i + 1;
 4:   while ( schedulable == TRUE and γ_i < N) do
 5:     γ_i = γ_i + 1;
 6:     /* check the schedulability of the affected task */
 7:     schedulable = is_task_schedulable(T_j);
 8:     if (schedulable == FALSE) then
 9:       γ_i = γ_i − 1;
10:     end if
11:     j = j + 1;
12:   end while
13:   schedulable = TRUE;
14: end for
15: return Γ;
```

This algorithm can be used for both fixed-priority as well as dynamic-priority schemes. The procedure *is_task_schedulable()* evaluates the schedulability of the system using either level-i busy period analysis for fixed-priority systems, or by computing the maximal blocking a task can endure without violating its schedulability for dynamic-priority schemes.

The MPTAA was analyzed and was shown to always find the maximal preemption-threshold assignment if one exists. This means the preemption-thresholds will be as large as possible. Since in our case we always start with a feasible assignment (the identity assignment), the maximal preemption-threshold assignment always exists (in the worst-case being equal to the identity assignment) and will always be found by the MPTAA.

PTS REDUCES NUMBER OF PREEMPTIONS

In 1999, Wang and Saksena evaluated how effectively PTS reduces the number of preemptions required to meet deadlines.[2] They found that PTS reduced preemptions by 5% to 32% when compared with fully-preemptive scheduling.

They evaluated randomly-generated periodic task sets. Each task set consists of a number of tasks, each with a computation time C_i and period T_i (which is also its deadline). The period is chosen randomly from the range 1 to *MaxPeriod* using a uniform probability distribution function. The computation time C_i is then defined with a uniform probability distribution function. This simulation approach allows us to see the behavior of different scheduling approaches across a range of workloads and evaluate their sensitivity to different factors.

Wang and Saksena measured the number of context switches occurring in a 100,000 time-unit execution for

Average percentage reduction in number of preemptions for PTS as compared with pure preemptive scheduling.

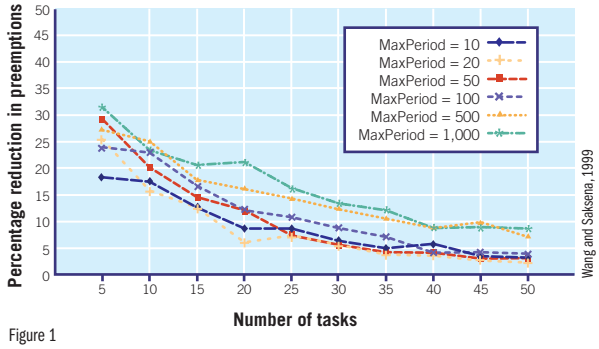


Figure 1

Number of context switches under PTS compared with preemptive scheduling.

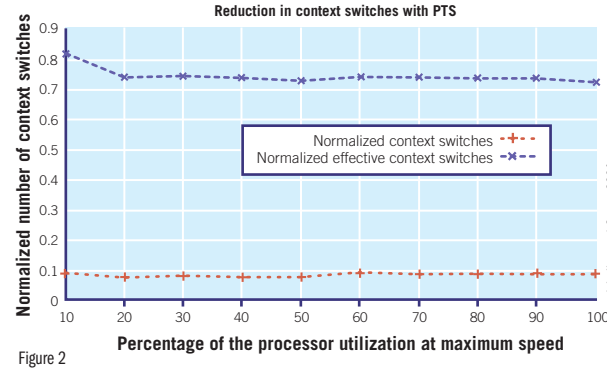


Figure 2

each of the workloads. The results are presented in **Figure 1** and show two interesting trends. PTS does reduce the number of preemptions significantly, depending on the number of tasks and the MaxPeriod parameter. First, the number of tasks in a workload matters because it determines how long the system runs before a task finishes, allowing another task to run without preempting the first task. For constant processor utilization, more tasks means the processor's busy time is broken into smaller pieces, reducing the time to wait. Second, the MaxPeriod parameter matters because a broader range of periods (and hence deadlines) provides more opportunities to limit preemptions while still meeting deadlines. A smaller range reduces preemption reduction because task deadlines are more closely synchronized.

In 2004, Jejurikar and Gupta evaluated using PTS in a cache-based system in conjunction with dynamic voltage scaling to reduce system energy requirements.[5] They found that PTS significantly reduced the number of context switches required. Randomly-generated periodic task sets were used, each with 10 to 20 tasks, and deadlines equaled periods. Periods ranged over a factor of 10x.

The authors counted two types of context switches (actual and effective) and compared them against those for a fully-preemptive scheduling approach. Effective context switches occur when a task begins or ends execution, and are important because they directly affect cache performance by disrupting the locality of memory references.

Jejurikar and Gupta in **Figure 2** showed that on average, PTS reduced the number of context switches by 90% from the fully-preemptive approach, essentially independent of processor utilization. The number of effective context switches was reduced by 19% to 29%, improving memory system performance by about the same amount.

PTS REDUCES STACK SPACE REQUIREMENTS

Threads in a nonpreemptive group can share stack space since their execution is not interleaved. Run-to-completion threads and some blocking threads qualify. Baker's Stack Resource Protocol calculates the possible maximum blocking times of tasks, allowing a system designer to determine whether deadlines can be met.[3] SRP involves modifying the scheduler to not start executing a task if its preemption level is not high enough. With this, the task will not block partway through, since it doesn't start running unless all resources are available.

My own research group and others have evaluated how PTS can reduce stack space requirements. We showed that PTS can be used with any scheduling algorithm (such as rate monotonic and earliest deadline first) and it will always render the smallest total stack space requirements[6] when using the maximal preemption-threshold assignment algorithm described by Wang and Saksena.[2] This further extends their finding that the algorithm minimizes the number of separate stacks required, which does not necessarily minimize total stack space. Gai extended SRP to support preemp-

Stack space required for PTS with a fixed-priority policy rises with both task period variability and system utilization.

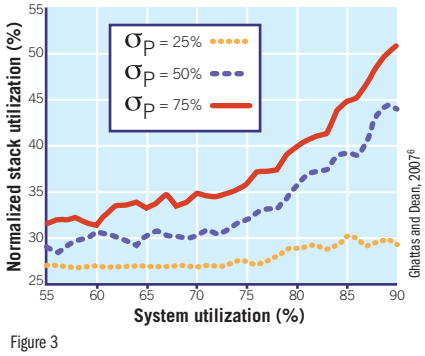


Figure 3

Stack space required for PTS with a dynamic-priority policy depends mostly on task period variability.

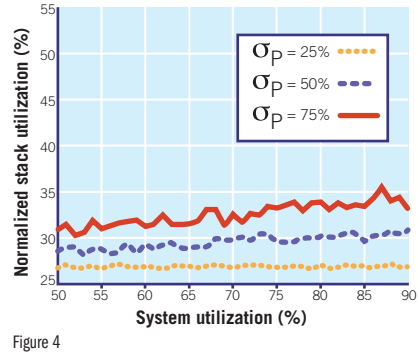


Figure 4

tion-thresholds and showed that it minimizes total stack space requirements.[4]

We investigated workload characteristics that affect the stack size reduction achievable through PTS. For example, the optimal stack utilization for some workloads through the use of PTS can be as small as 20% of the stack space utilization required by the fully-preemptive version of the system, while others need 80%. We simulated and analyzed 40,000 randomly-generated systems of 10 threads each to better understand PTS. The results shown in **Figure 3** are summarized below.

We first investigated the effect of the system utilization on the optimal stack utilization required with PTS. Using the MPTAA, the optimal stack space required by each system was computed and normalized to the stack space required by the fully-preemptive version of the system. The average normalized stack utilizations were then plotted as a function of the overall system utilization and the standard deviation in the task periods. The results are shown in **Figures 3 and 4** for the fixed-priority and the dynamic-priority schemes, respectively.

First, consider the fixed-priority scheme in Figure 3. At low utilizations, the system's stack space requirements might be less than 30% of those of a fully-preemptive system. However, at higher utilization levels, the variation in the tasks' periods increasingly affects the savings attainable. With $\sigma_P = 25\%$ the savings are only slightly dependent on the utilization level.

On the other hand, as the variance and standard deviation of the period increases, the savings attainable decrease significantly at higher utilizations. This is because task WCETs are larger so it is much harder to maintain the system's schedulability while minimizing preemptions. This becomes very apparent at high system utilizations where there is far less slack time.

Second, consider the dynamic-priority scheme in Figure 4. The normalized stack space utilizations in this case are much more uniform across all utilization levels. This is because the dynamic priority scheme (EDF in this case) is much more adaptive to the workload characteristics and has a higher schedulable utilization than a fixed-priority scheme such as RM. This more-efficient scheduling allows more preemption limiting to occur before schedulability is lost.

Another interesting property is the distribution of the 40,000 systems among the different normalized stack space utilization levels. To evaluate this property, we constructed histograms showing the percentage of systems versus the normalized (optimal) stack space utilization achievable by each system. **Figures 5 and 6** show this distribution for the overall system utilization levels of 60%, 70%, 80%, and 90%, respectively. Again we can see that the workloads scheduled with the dynamic-priority schemes depend less on the system utilization level than those in a fixed-priority scheme.

In Part 2 of this series online at Embedded.com, PTS offers advantages in the schedulability of real-time systems while affording reduced overhead, and other benefits as well. ■

Alexander G. Dean, PhD, is associate professor in the Department of Electrical and Computer Engineering at North Carolina State University and a member of its Center for Efficient, Scalable and Reliable Computing. He received his BS EE from the University of Wisconsin and MS and PhD ECE degrees at Carnegie Mellon. His research focus in embedded systems includes compiling for concurrency and performance, energy efficient use of COTS processors, memory allocation, and benchmarking for real-time systems and robust embedded system design.

ENDNOTES:

1. Preemption-Threshold is a trademark of Express Logic, Inc.
2. Wang, Yun and Manas Saksena. "Scheduling fixed-priority tasks with preemption threshold," *Real-Time Computing Systems and Applications*, 1999. RTCSA '99. *http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=811269&isnumber=17590.*
3. Baker, T. P. "Stack-based scheduling of real-time processes," *Real-Time Systems Symposium, 1990. Proceedings., 11th. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=128747&isnumber=3599.*
4. Gai, Paolo, Giuseppe Lipari, and Marco Di Natale. "Minimizing Memory Utilization of Real-Time Task Sets in Single and Multi-Processor Systems-on-a-Chip," *IEEE International*, 22nd IEEE Real-Time Systems Symposium (RTSS'01), 2001. *http://doi.ieeecomputersociety.org/10.1109/REAL.2001.990598*
5. Jejurikar, Ravindra and Rajesh Gupta. "Integrating Preemption Threshold Scheduling and Dynamic Voltage Scaling for Energy Efficient Real-Time Systems," *Proceedings of the Ninth International Conference on Real-Time Computing Systems and Applications*, 2004. *http://dar.ucsd.edu/site/pubs/jerjurikar-rtcsa04.pdf*
6. Ghattas, Rony and Alexander G. Dean. "Preemption Threshold Scheduling: Stack Optimality, Enhancements and Analysis," *Proceedings of the 13th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2007)*, IEEE, 2007. A version of this article is at *www.cesr.ncsu.edu/agdean/Papers/ghattas-Preemption.pdf.*

PTS and a fixed-priority policy dramatically reduce the stack space required, even for high-utilization systems.

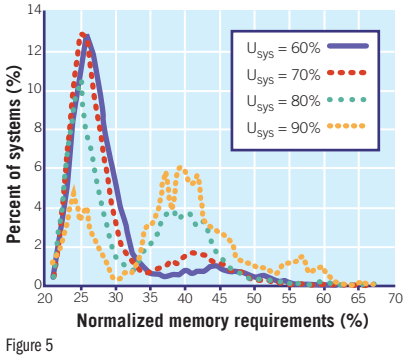


Figure 5

PTS and a dynamic-priority policy reduce the stack space required even better than a fixed-priority policy.

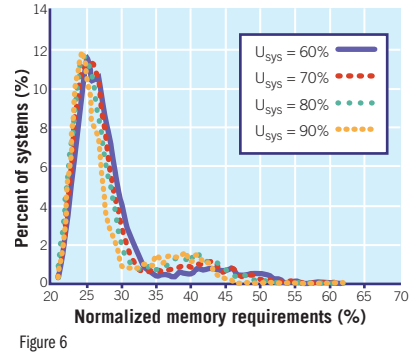


Figure 6

programmer's toolbox

from page 13

get a running value of V and therefore of σ.

As before, it's almost easier to show you the code than it is to describe the process. See **Listing 2**.

Next, look at **Figure 3**, a graph like Figure 2 only you see the sequential values of the average and the error band, and the way they evolve as new data comes in.

Perhaps you've seen graphs with error bars showing the expected deviations from each measurement. That's what I'm trying to display here, only Microsoft Excel won't let me show error bars. So the large black dashes will have to do.

CAVEAT CALCULATOR

I need to mention one or two practical problems. Take a look at the running sums in Equations 10 and 21. See any potential problems?

That's right. The two sums can grow without limit. If you process the input signal at a high enough rate, and for long enough, sooner or later both sums are going to overflow. Presumably, the sum of the squares is going to overflow first.

How long will it take for your filter to overflow? That's 10^{287} times the current age of the universe! This range is head and shoulders different from, say, the Y2K problem. It's safe to say we won't see the sums overflow in our lifetimes.

We can do certain things to alleviate the problem. First, note carefully that the measurements y_i can have units. If you're measuring the distance to Alpha Centauri and using units of millimeters, you're going to have overflow problems a lot sooner. One thing you can do is to choose your units more carefully and normalize them to some standard value—Astronomical Unit (AU), for example.

Or you might split up the computation into stages. At some point, where the sums are starting to get out of hand, you might just store away all the values to date and start a new set of computations.

The easiest thing to do is to normalize all the numbers by the number of samples. That is, maintain running averages rather than running sums. We can do that easily, as shown in Equation 9.

Interestingly enough, one approach to the problem is to ignore it. This potential for overflow is an excellent argument for computing everything in floating point, even in small, integer-only computers. Better yet, compute in double precision, where the maximum value is a staggering 10^{308}.

How big is that? Imagine that you're sampling a signal at the rather high (for embedded systems) rate of 1000 Hz. Imagine also that you've chosen your units wisely, so that all the measurements are of order 1. How long will it take for your filter to overflow?

It's $10^{308} \div 1000 = 10^{305}$ sec $= 3 \times 10^{297}$yr.

That's 10^{287} times the current age of the universe!

This range is head and shoulders different from, say, the Y2K problem. The range of a double precision number is so staggeringly large, I think it's safe to say we won't see the sums overflow in our lifetimes.

So one acceptable solution to the overflow problem is simply to ignore it. As long as you're using floating point—or even better, double precision—arithmetic, the overflow is simply not going to happen.

Another problem in the calculation is more challenging. It lies in Equation 20. Suppose that the average value of the data set, $\mathbf{y}$, are reasonably large, but the noise is very small. The residuals in Equation 12 are all going to be small, as is their sum in Equation 13.

But we aren't summing residuals in Equation 20—we're summing the squares of the measurements themselves. This sum can be a whole lot larger. For that reason, we can end up, in Equation 20, subtracting nearly equal numbers and thereby introducing a loss of precision.

There is no quick cure for this problem. If you need the benefits of sequential processing, you'll have to accept that your end result may not be as accurate as you might like.

You can take quite a number of steps to fix the problem, but none are straightforward, and all are very much problem dependent. You might choose, for example, to reference the measurement to some expected value, other than zero. In that Alpha Centauri example, a baseline of four light-years might help.

Like most problems, this one can be solved but don't expect your solution to one problem to work in all possible other problems. In the real world of processing real-world signals, you can expect to tailor the solution to the problem. How surprising is *that*?

COMING SOON

This month, I've shown you how to calculate the mean, variance, and standard deviation of a set of measurements. I showed you how to do it the classical way, processing all the data in batch mode, and I've shown you the tricks to do the same calculation sequentially, as the data comes in. I studiously avoided talking about statistics, so we clearly have more things to talk about. I'll do that next month. See you then. ■

By Jack G. Ganssle

Watchdogs redux

Nearly a decade ago I wrote a series of three articles about watchdog timers (WDT). It was my contention at the time that most WDTs were poorly designed. Too many still are.[1]

I won't repeat my arguments here since they're available online on Embedded.com (*www.eetimes.com/4024495*). However, I stated that a WDT is the last line of defense against product failure. Designed correctly, the system is guaranteed to recover from a crash; anything else may result in a ticked-off customer. Or loss of an expensive mission. Or, for the unfortunate users, injury and death.

Remember that when a program crashes, it generally runs off and starts executing code from some random address. Rarely does the application actually stop; if it does stop, that's usually only after it executes a lot of incorrect instructions.

So what's the state of the art today? A complete survey is impossible, but here are a few data points.

Texas Intruments' MSP430 family is composed of a wide range of nifty very low power 16-bit microcontrollers. The documentation shows a very impressive-looking WDT block diagram (see Figure 13-1 in TI's PDF), but the reality is less thrilling.[2] At any time, the code can turn the protection mechanism off, which means a crashed program running rogue code can issue an instruction

Vendors and researchers are trying some new tricks with watchdog timers. Here are some notable attempts to improve the old dog.

that disables the WDT. The system crashes and never recovers.

The MSP430's instruction set is refreshingly simple, generally using a single word to represent, for instance, a MOV instruction. A nice feature is that to change any WDT setting one uses a MOV with the upper byte set to 0x5a; the lower 8 bits contain the command. Try to write to it with anything else in the MSB and the system will reset. The lower bits have a variety of configuration information that disables/enables the WDT, selects the clock source, or even switches the WDT to act as a simple timer.

What are the odds that crashed code will issue exactly a 0x5a in the upper byte? Pretty slim, of course. But not zero, and probably a lot less than zero. A lot of move instructions will be in the code, and some will be followed by an ADD, which is what 0x5a represents. Much better would be a configuration that allows one to access the WDT configuration only once. Or perhaps only once after resuming from a low-power mode.

Then there's Freescale's newish 32-bit Coldfire+ line, like the MCF51Qx.[3] Instead of "watchdog," Freescale prefers the awkward phrase "Computer Acting Properly" (COP). But it does offer a very intriguing feature. In general, one pets the watchdog, uh, COP, by writing 0x55 and then 0xaa to the control register. But in one mode, that sequence must be sent in the last 25% of the COP timeout period. A premature write results in a reset. Odds of an errant program getting the timing Goldilocks-correct (not too often, nor too infrequently) are tiny.

The part also generates a reset if any attempt is made to execute an illegal instruction. That's somewhat different from most CPUs, which issue an illegal op-code interrupt. I rather like Freescale's approach, since interrupt handlers are not guaranteed to work if the code crashes. A blown stack, corrupt PC (on some CPUs if the PC is odd, a fault is taken), or a vector base register changes. This also

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.

EE Times Virtual Conference

Upcoming Virtual Conferences

EE Times virtual event
MULTICORE

When: Thurs., Mar. 24, 2011, 11am – 6pm EDT

On Demand Virtual Conferences

Integrating Touch Interfaces
www.eetimes.com/touch

STMicroelectronics Virtual Conference: Robust Ecosystem = Embedded Succes
www.eetimes.com/mcu

System-on-chip 2.0
www.eetimes.com/soc

Embedded Linux
www.eetimes.com/linux

Approaching Multicore
www.eetimes.com/multicore

Advances in Power Management
www.eetimes.com/power

EE Times, the leading resource for design decision makers in the electronics industry brings to you a series of Virtual Conferences. These fully interactive events incorporate online learning, active movement in and out of exhibit booths and sessions, vendor presentations and more. Because the conference is virtual you can experience it from the comfort of your own desk. So you can get right to the industry information and solutions you seek.

Why you should attend:

- Learn from top industry speakers
- Participate in educational sessions in real time
- Easy access to EE Times library of resources
- Interact with experts and vendors at the Virtual Expo Floor
- Find design solutions for your business

For sponsorship information, please contact:
Christian Fahlen, 415-947-6623 or
christian.fahlen@ubm.com

UBM Electronics

suggests that it's a good idea to fill unused flash at link time with an illegal op code, and on power-up fill all of RAM with a similar instruction, so that errant code waltzing through memory is likely to generate a reset.

Another nice touch is that the reset pin is open drain and is asserted when any of these errors occur. Tie it to the peripheral reset inputs. Even if wandering code issues output instructions their potentially scrambled little brains will be straightened out.

STMicroelectronics has a line of Cortex-M3 devices. The M3 has become extremely popular for lower-end embedded devices, and ST's STM32F is representative of these parts (although the WDT is an ST add-on and does not necessarily mirror other vendors' implementations). The STM32F has two different protection mechanisms. An "Independent Watchdog" is a pretty vanilla design that has little going for it other than ease of use. But their Window Watchdog offers more robust protection. When a countdown timer expires, a reset is generated, which can be impeded by reloading the timer. Nothing special there. But if the reload happens too quickly, the system will also reset. In this case "too quickly" is determined by a value one programs into a control register.

Another cool feature: it can generate an interrupt just before resetting. Write a bit of code to snag the interrupt and you can take some action to, for instance, put the system in a safe state or to snapshot data for debugging purposes. ST suggests using the interrupt service routine (ISR) to reload the watchdog—that is, kick the dog so a reset does not occur. Don't take their advice. If the program crashes, the interrupt handlers may very well continue to function normally. And using an ISR to reload the WDT invalidates the entire reason for a window watchdog.

The WDT cannot be disabled once enabled—good thinking, folks! But oddly, the other configuration registers can be changed at will, which can make the watchdog behave incorrectly.

The WDT cannot be disabled once enabled—good thinking, folks! But oddly, the other configuration registers can be changed at will.

A NOVEL APPROACH

The latest issue of *IEEE Embedded Systems Letters* has an article that practically grabbed me by the throat.[4] Titled "Control Focused Soft Error Detection for Embedded Applications" by Karthik Shankar and Roman Lysecky, it's a bit academic and rather a slog to get through. But the authors have come up with a fascinating twist on the concept of watchdog timers. In fact, they don't use the word "watchdog," and no timers are involved.

The idea is quite simple and at first glance not particularly novel: monitor the addresses the processor issues and compare those against a profile obtained during development. If the CPU goes to an unexpected location, take some sort of remedial action. Do the same if it doesn't issue an expected address. The authors go further and compare against the number of expected loop iterations and the like.

But what got my attention is how they monitor addresses. They simply suck in compressed trace data from the processor's serial trace data port. The paper talks about using an ARM CPU, but other parts also support various kinds of serial trace data, and there's even a standard named Nexus-5001, which is IEEE–ISTO 5001–2003.[5]

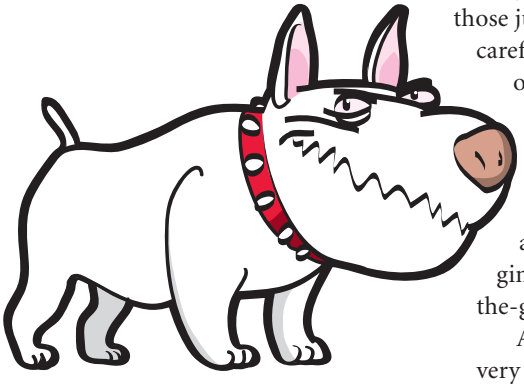
ARM supplies a bewildering array of debugging IP, including a macrocell that sends trace data out just two pins. The so-called Program Flow Trace (PFT) is described at ARM's infocenter.[6] It can be set up in a zillion different configurations, but will at the very least emit compressed information that lets one track changes in program flow. For instance, the execution of a branch instruction pushes address data out. So does Branch with Link, which is ARM's way of calling a subroutine. Similar instructions in Thumb mode do the same.

There's a lot to like about the Shankar and Lysecky's approach. First, modern processors are barricaded behind caches, pipelines, and speculative execution logic. Even if you're using a CPU that has address pins (as opposed to a self-contained microcontroller or IP on an FPGA), those address lines simply don't match anything that is going on in the processor's grey cells.

Second, pins are valuable. By squeezing addresses through a couple of debug pins few of these valuable resources are consumed. And logic is cheaper than pins; adding circuitry to decompress the trace stream and make sense of it may cost very little. In the paper, the authors state that the overhead for their particular approach was only 3% of the silicon area of an ARM1156T.

Third, most watchdogs use ad hoc approaches that just are not reliable indicators of system operation. This new approach lets the designer decide just how fine-grained the monitoring should be. Check an occasional call… or watch every instance of every branch and call.

There's no overhead in the system's firmware. Zilch. Unlike traditional watchdog approaches, address profiling is transparent to the software.



Fourth, there's no overhead in the system's firmware. Zilch. Unlike traditional watchdog approaches, which require one to seed the code with instructions to periodically kick the dog to keep it from resetting the system, address profiling is transparent to the software.

There are a few caveats, of course. The logic needed to make sense of the address information is substantial, and is probably impractical unless implemented in an FPGA. Building such a monitor would be a lot of work. But it needs to be done once, and then can ever after be used in a succession of products. I can see this as packaged IP sold by a third party, or as an open source project.

Remember, though, there's no guarantee that your ARM CPU will have the trace logic needed. Every vendor is free to include the IP or not.

GOING DEEPER

I haven't thought out all of the implications or possible ways to actually use this idea in a real embedded system, but here are a few ideas.

One problem with the proposed approach is that every recompile will shift all of the system's addresses, requiring the developer to re-profile the code to determine where the branches are. An alternative is to monitor just function calls, but use a level of indirection. Build a table of jump instructions that lives at a fixed address in memory; each entry corresponds to a particular function. Make calls through that table. Then monitor those jumps. One would have to be careful that the compiler didn't optimize the jumps away.

This does mean the software changes a bit, of course. But more than a few of us use these jump tables anyway. They can aid debugging and sometimes simplify on-the-go firmware updating.

ARM's program flow trace also very intriguingly sends address information out when a DSB, DMB, or ISB instruction is executed. DSB (Data Synchronization Barrier) holds up program flow until all of the instructions before it complete; DMB (Data Memory Barrier) ensures that memory accesses before it compete before one after it starts, and ISB (Instruction Synchronization Barrier) flushes the instruction pipeline, ensuring that the instructions after the ISB are fetched from memory or the cache.

Why are these interesting instructions? ARM mandates that at the very least a DMB be issued before locking or unlocking a mutex, and before incrementing or decrementing a semaphore. One could monitor these, just like watching branches, to ensure that the code is properly going through all of the activities mediated by the RTOS. In fact, monitoring only these actions may be simpler than the addresses, because program flow can change quite a lot even from a simple change (requiring re-profiling), but resource management tends to change much less frequently.

As far as the DSB and ISB instructions, I'm not quite sure how they could offer useful watchdog information, but something makes me think they could offer some interesting options.

PARTING THOUGHTS

If you're building an electronic toothbrush, watchdogs are probably not terribly important. But an automated reset helps boost consumer confidence in our products' quality. Everyone hates the "remove batteries and wait 30 seconds" dance.

Many vendors are putting more thought into their WDT designs; some are doing a pretty good job. But we have a long way to go, and the wise developer will apply sound engineering practices to this often-neglected part of the system.

The article I cited shows that some ingenious approaches are being used. Consider adding a bit of hardware support if robustness is an important requirement. ■

ENDNOTES:

1. Ganssle, Jack. "Born to Fail" Embedded.com, December 12, 2002. Available at *www.eetimes.com/4024495.*
2. Texas Instruments. "MSP430x5xx/ MSP430x6xx Family User's Guide" Literature Number: SLAU208H, June 2008–revised December 2010. Available at *http://focus.ti.com/lit/ug/slau208h/slau208h.pdf*
3. Freescale Semiconductor. "MCF51QM128 Reference Manual: Supports the MCF51QM32, MCF51QM64, and MCF51QM128," Document Number: MCF51QM128RM, Rev. 0, November 11, 2010. *http://cache.freescale.com/files/32bit/doc/ref_manual/MCF51QM128RM.pdf*
4. Shankar, Karthik and Roman Lysecky. "Control Focused Soft Error Detection for Embedded Applications," *IEEE Embedded Systems Letters,* December 2010, Volume 2 Number 4.
5. The Nexus 5001 Forum—A Program of the IEEE–ISTO. See *http://www.nexus5001.org.*
6. ARM. "CoreSight Program Flow Trace Architecture Specification, v1.0." ARM, 2008. Available at *http://infocenter.arm.com/help/topic/com.arm.doc.ihi0035a/IHI0035A_coresight_pft_architecture_spec.pdf.*

AS EMBEDDED TECHNOLOGY ADVANCES WE'RE RIGHT THERE WITH YOU

A N D R O I D ▪ N U C L E U S ▪ L I N U X

MENTOR EMBEDDED. Innovation is boundless. Mastering its possibilities requires a partner with the very latest in embedded software solutions. Mentor Embedded delivers solutions that reduce your development risk and shorten project cycles. We provide proven software and tools, along with services for Linux and Android development that enable our customers to succeed in emerging areas such as Android beyond mobile, multi-OS, multicore, and advanced 3D user interfaces. As the industry's leading independent vendor, Mentor Embedded stands ready for your next innovation. To learn more visit **www.mentor.com/embedded.**

Mentor Graphics®

Who makes the fastest real-time oscilloscopes?

20 MHz - 40 MHz

100 MHz - 200 MHz

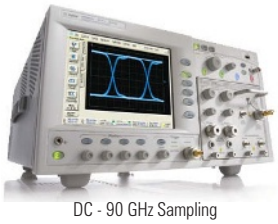
100 MHz - 500 MHz

100 MHz - 1 GHz

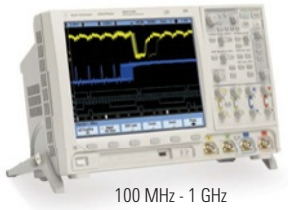
60 MHz - 200 MHz

100 MHz - 1 GHz

The fastest-growing oscilloscope company.*



DC - 90 GHz Sampling



100 MHz - 1 GHz

2.5 GHz - 13 GHz

600 MHz - 4 GHz

Introducing 16-32 GHz
Agilent Infiniium 90000 X-Series

Our portfolio offers families engineered to deliver the best:

- Best measurement accuracy
- Broadest measurement capability
- Best signal visibility
- More scope than you thought you could afford

Are you using the best scope?
Take the 5-minute scope challenge and find out.
www.agilent.com/find/scopecheck

**Prime Data 2009 Market Growth Analysis.*

u.s. 1-800-829-4444 canada 1-877-894-4414

Agilent Technologies