CMP
United Business Media

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

VOLUME 20, NUMBER 12

DECEMBER 2007

Embedded Systems Design

Learn today • Design tomorrow

HI-TECH C® PRO for the PIC10/12/16 MCU Family

ANSI C Compiler comes with Omniscient Code Generation™

DENSER CODE in ½ the time

HI-TECH C PRO for the PIC10/12/16 MCU Family - Based on the reliable PICC™ STD compiler, and supporting Microchip's PIC10/12/14/16/17 series of MCUs, the new *HI-TECH C PRO for the PIC10/12/16 MCU Family* ANSI C Compiler comes with Omniscient Code Generation.

OMNISCIENT CODE GENERATION
Extracts information from multiple source files simultaneously, allowing more intelligent state-of-the-art code generation that:

- Automatically handles memory banking without requiring special qualifiers;
- Optimizes the size of each pointer variable in your code based on its usage;
- Eliminates the need for many non-standard C qualifiers and compiler options;
- Produces more optimal interrupt context switching code;
- Customizes the functionality of the included `printf` library function;
- Automatically analyzes user assembly and object code files; and
- Is simple to use.

DENSER CODE, IN HALF THE TIME.
Code compiled with *HI-TECH C PRO for the PIC10/12/16 MCU Family* can deliver up to 30% denser code than you are currently producing. Additionally, automatic memory and stack management could cut your development time in half.

FREE 45 DAY EVALUATION
A fully functional 45 day trial version of the *HI-TECH C PRO for the PIC10/12/16 MCU Family* will be available, free of charge from HI-TECH's website. Order your demo at www.htsoft.com/portal/ccpiccpro.

PRICE AND AVAILABILITY
HI-TECH C PRO for the PIC10/12/16 MCU Family includes, at no extra cost:

HI-TECH Priority Access™ - 12 months access to updates and technical support; and

HI-TECH Satisfaction Guarantee - a hassle-free 30 day money back guarantee.

FOR RELEASE THIS MONTH

HI-TECH SOFTWARE

Ph: 800 735 5715 **Web:** microchip.htsoft.com

HI-TECH Software proudly supports the Microchip brand with industrial-strength software development tools and C compilers. PICC is licensed exclusively to HI-TECH Software by Microchip Technology Inc.

Recognized Microchip Third-Party Tool Provider

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

Embedded Systems Design

www.embedded.com

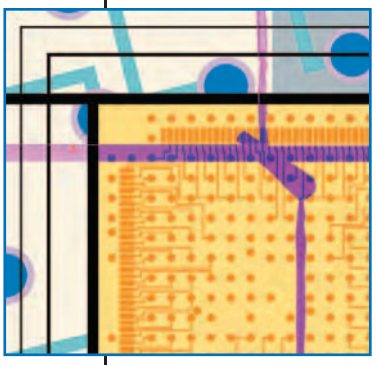
DECEMBER 2007 VOLUME 20, NUMBER 12

Cover Feature: Employ the proper flash memory in your design

BY VIJAY K. DEVADIGA

To NAND or NOR. That is the question. Different applications and functions should be handled with different types of flash memory.

«18

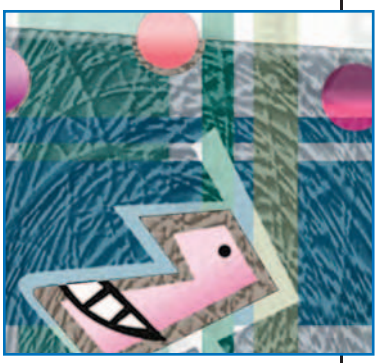


Lower the cost of intelligent power control with FPGAs

26»

BY WENDY LOCKHART

Combining a programmable solution with an industry-standard processor core can save time, money, and real estate.



The basics of embedded multitasking on a PIC

BY GAMAL ALI LABIB

High-end concepts typically aren't applied to an 8-bit MCU. But don't rule them out just yet.

«34

Do-it-yourself embedded Linux development tools

42»

BY ALEXANDER SIROTKIN

These key tools can be fine-tuned to fit your needs and your style of work.

EMBEDDED SYSTEMS DESIGN (ISSN 1558-2493 print; ISSN 1558-2507 PDF-electronic) is published monthly by CMP Media LLC., 600 Harrison Street, 5th floor, San Francisco, CA 94107, (415) 947-6000. Please direct advertising and editorial inquiries to this address.
SUBSCRIPTION RATE for the United States is $55 for 12 issues. Canadian/Mexican orders must be accompanied by payment in U.S. funds with additional postage of $6 per year. All other foreign subscriptions must be prepaid in U.S. funds with additional postage of $15 per year for surface mail and $40 per year for airmail. POSTMASTER: Send all changes to EMBEDDED SYSTEMS DESIGN, P.O. Box 3404, Northbrook, IL 60065-9468. For customer service, telephone toll-free (877) 676-9745. Please allow four to six weeks for change of address to take effect. Periodicals postage paid at San Francisco, CA and additional mailing offices. EMBEDDED SYSTEMS DESIGN is a registered trademark owned by the parent company, CMP Media LLC.

BPA INTERNATIONAL

columns

departments

in person

ESC Silicon Valley

San Jose Convention Center
April 14–18, 2008
www.embedded.com/esc/sv/

on-line

www.embedded.com

Web archive:
www.embedded.com/archive

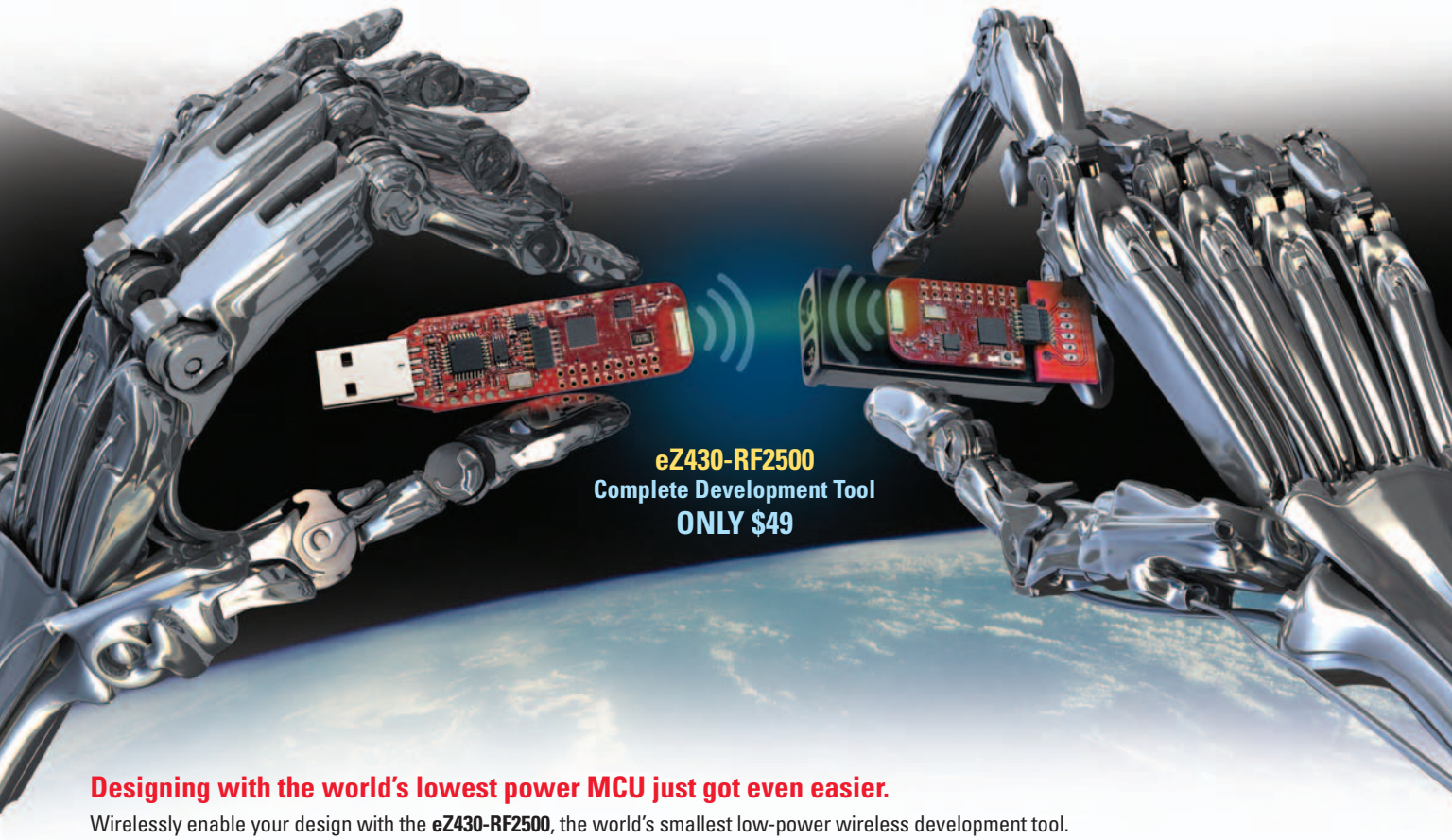
Article submissions:
www.embedded.com/wriguide

FAST-FORWARD TURNAROUND TIME.

The pursuit of perfection takes time. But you can speed up the unnecessary slowdowns. Windows® Embedded helps accelerate the creation of next-generation smart, connected devices by giving you access to an active global community: forums, newsgroups, tutorials, support, and the MSDN® Windows Embedded Developer Center. At the same time, Windows Embedded Partners are there to provide expert support and services to help get your devices delivered on time. Learn more about how to fast-forward device development at: **windowsembedded.com/fastforward**

Windows® Embedded

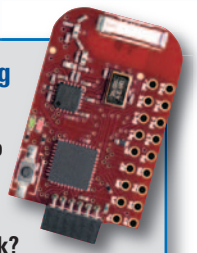
MSP430 Goes Wireless



Designing with the world's lowest power MCU just got even easier.

Wirelessly enable your design with the **eZ430-RF2500**, the world's smallest low-power wireless development tool. At **only $49**, the tool includes a USB emulator to program and debug your application in-system and two 2.4-GHz wireless target boards featuring the highly integrated MSP430F2274 ultra-low-power MCU. All the required software is included such as a complete Integrated Development Environment and SimpliciTI™, a propriety low-power star network stack, enabling robust wireless networks out of the box. The MSP430F22x4 combines 16-MIPS performance with a 200-ksps 10-bit ADC and 2 op-amps and is paired with the CC2500 multi-channel RF transceiver designed for low-power wireless applications.

MCU: MSP430F2274		RF Transceiver: CC2500	
Flash	32 KB	Frequency	2.4 GHz
RAM	1 KB	Sensitivity	–101 dBm
Analog	10-bit ADC, 2 op amps	Max Data Rate	500 kbps
Communication	UART/LIN, IrDA, (2) SPI, I²C	RX Current	13.3 mA
Standby Current	0.7 µA	TX Current	21.2 mA



Coming Soon!

Need to extend your network?

Additional eZ430-RF2500T target boards can be ordered for $20 each.

www.ti.com/ez430-rf

Order Today! **www.ti.com/ez430-rf or 800.477.8924, ext. 4037**

1880A0

Technology for Innovators™

Texas Instruments

Embedded Systems Design

Editor in Chief
Richard Nass
(201) 288-1904
rnass@cmp.com

Managing Editor
Susan Rambo
srambo@cmp.com

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

Art Director
Debee Rommel
drommel@cmp.com

European Correspondent
Colin Holland
colin.holland@btinternet.com

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

Production Manager
Pete Scibilia
pscibili@cmp.com

Director of Audiences Services
Kristi Cunningham
kcunningham@cmp.com

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

Back Issues
Kelly Minihan
(800) 444-4881 (toll free),
Fax: (785) 838-7566

Article Reprints, E-prints, and Permissions
PARS International Corp.
102 West 38th Street, Sixth Floor
New York, NY 10018
(212) 221-9595, Fax: (212) 221-9195
reprints@parsintl.com
www.magreprints.com.quickquote.asp

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

#include

BY Richard Nass

20 years of Embedded

The year 2008 marks two important milestones for our Embedded franchise, which consists Embedded.com, this magazine, and the Embedded Systems Conference. In April, we will hold the 20th Annual California-based Embedded Systems Conference. And then later in the year, November to be exact, we'll mark 20 years of publishing *Embedded Systems Design* magazine (although it was known as *Embedded Systems Programming* for its first 17 years).

There's a common phrase that definitely applies here: the more things change, the more they stay the same. Back in 1988, 8-bit microcontrollers were shipping in huge volumes, clearly dominating the micro scene in terms of units shipped. Today, while the 8-bit micros themselves have changed dramatically, they're still the leader, by a wide margin. And some of those same cores, like the 8051, are still in use.

Throughout 2008, we'll be sprinkling in various items to commemorate our 20-year anniversary. For example, in each issue, we'll reprint one item that appeared 20 years ago. But it's not for nostalgic reasons. It's because that information is as relevant today as it was 20 years ago. While most of the articles we publish are available on Embedded.com, the articles from 20 years ago are not. We didn't start posting articles until 1996.

In addition, we'll be looking for information from our readers. For example, what do you consider the most important achievement in the embedded space in the last 20 years? Who has been the most influential individual? Software vendor? Hardware vendor? Do you have an anecdotal story relating to your job in the embedded industry?

What's the most significant thing that's happened to you at an Embedded Systems Conference (ESC) over the last 20 years? How many ESCs have you attended? Can you recall a career changing moment that occurred at an ESC?

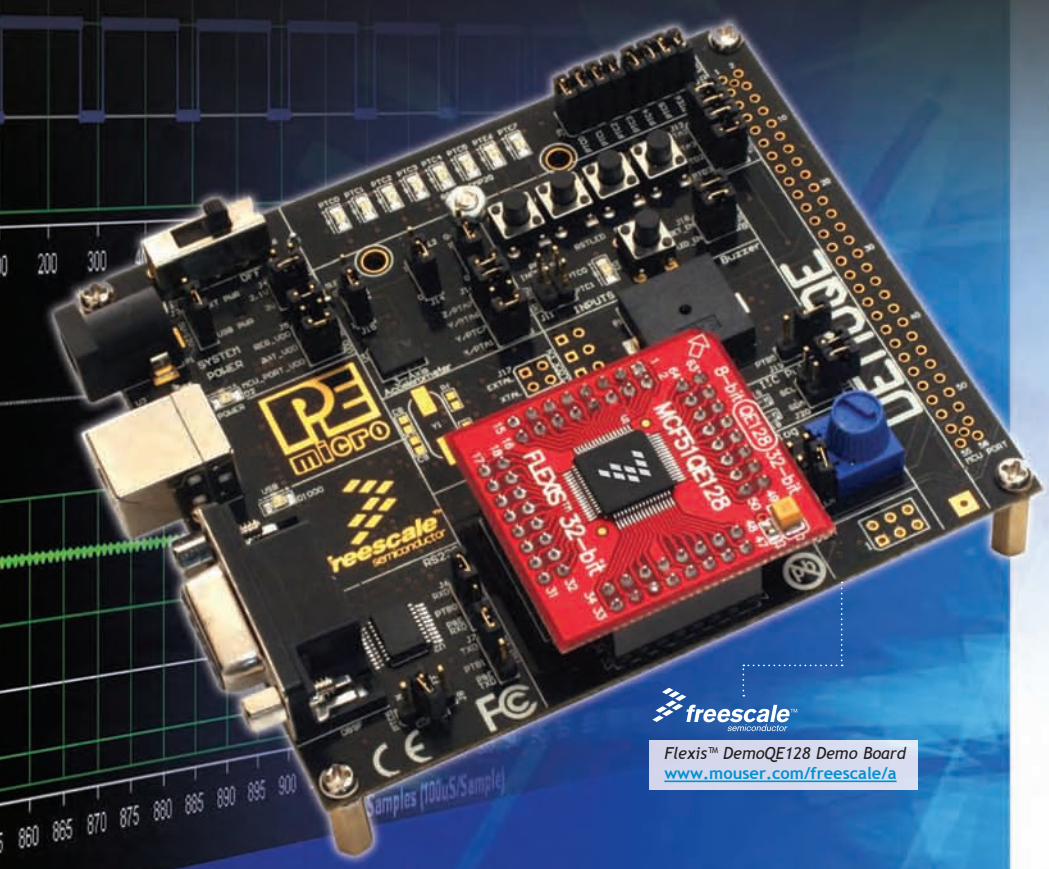
We want to hear from you. Go into the Forums section on Embedded.com and let us hear from you. In addition, you'll be hearing stories from the folks on our team, discussing some of the most memorable "embedded" moments that have occurred over the past 20 years. We'll also be taking a peek into the future, giving you a clue as to what you can expect to see in the embedded space over the next 20 years.

Richard Nass, rnass@cmp.com

Richard Nass is editor in chief of Embedded Systems Design. *You can reach him at rnass@cmp.com.*

The Newest Embedded Technologies

Network, USB, ZigBee®, WLAN, WiFi, RFID, Bluetooth™, GPS, and Embedded Control.



freescale semiconductor

Flexis™ DemoQE128 Demo Board
www.mouser.com/freescale/a

New Products from:

VDIP1 Module
www.mouser.com/ftdi/a

FTDI Chip

RCM4000
RabbitCore® Modules
www.mouser.com/rabbitsemi/a

RABBIT Semiconductor

MatchPort™ b/g
Embedded Device Server
www.mouser.com/lantronix/a

LANTRONIX®

The ONLY New Catalog Every 90 Days

Experience Mouser's time-to-market advantage with no minimums and same-day shipping of the newest products from more than 335 leading suppliers.

MOUSER ELECTRONICS®
a tti company

The Newest Products For Your Newest Designs

www.mouser.com
Over 890,000 Products Online

(800) 346-6873

Don't forget the APIs

In the article, "Use an MCU's low-power modes in foreground/background systems," (*Miro Samek, October 2007, article ID: 202103425*) the Atmel AVR microprocessor was reviewed and example code for the WinAVR toolchain (GNU) was given.

I was disappointed to see that the author seemed to know nothing about the APIs that are available from AVR-LibC, the C library for the WinAVR (GNU) toolchain. The example code given in the article was inline assembly statements, which are ultimately not needed. The AVR-LibC documentation can be found online at *www.nongnu.org/avr-libc/user-manual/*. AVR-LibC provides an Interrupt API and a Sleep API, which provides an excellent example of usage and description *www.nongnu.org/avr-libc/user-manual/group__avr__sleep.html*:

```
#include <avr/interrupt.h>
#include <avr/sleep.h>

...
cli();
if (some_condition) {
sleep_enable();
sei();
sleep_cpu();
sleep_disable();
}
sei();
```

"This sequence ensures an atomic test of some_condition with interrupts being disabled. If the condition is met, sleep mode will be prepared, and the SLEEP instruction will be scheduled immediately after an SEI instruction. As the instruction right after the SEI is guaranteed to be executed before an interrupt could trigger, it is sure the device will really be put to sleep."*

An article that just skims the surface of the issue of low-power settings in micros does not do justice to what hardware and software features are truly available to the end-user.

—Eric Weddington
Product Manager, Atmel
Creator of WinAVR

**Note: You can read all of Eric Weddington's comments at the end of the Samek article online.*

The assumption that changes to the M16C interrupt flag are effective immediately is incorrect. Unless the flag is changed by an RETI instruction the changed status takes effect beginning with the next instruction; in other words, it works the same as the AVR.

This information is contained in the M16C Software Manual. It is important that those writing code for the M16C also read through all the technical notes available on the Renesas website. These contain a lot of extremely important information especially about interrupts and power management.

— Andrew Geard
Senior Software Engineer
(online comment)

Quine-McClusky algorithm

The Quine-McClusky is NOT a reasonable algorithm for logic reduction.

Based on this article (*Jack Crenshaw, "All about Quine-McClusky," article ID: 29111968*), I implemented the algorithm and it turns out that the required memory and processing time explode if you use more than four or five variables. So I went back to the drawing board and designed a smarter algorithm that beats the pants off of QM. The new algorithm can easily handle 20+ variables and reduce the table in a few milliseconds.

While I appreciate the spirit of the article, the Quine-McClusky was nothing but a waste of my time.

—Damon Bohls
Software Engineer
Austin, TX

Jack Crenshaw responds: *The focus of my columns has never been to provide canned software or even canned algorithms. Rather, it's to educate. Even if I'm using a commercial tool, I still feel better if I know what it's doing inside. I feel better yet, knowing I could solve small problems by hand, if necessary. Most readers agree.*

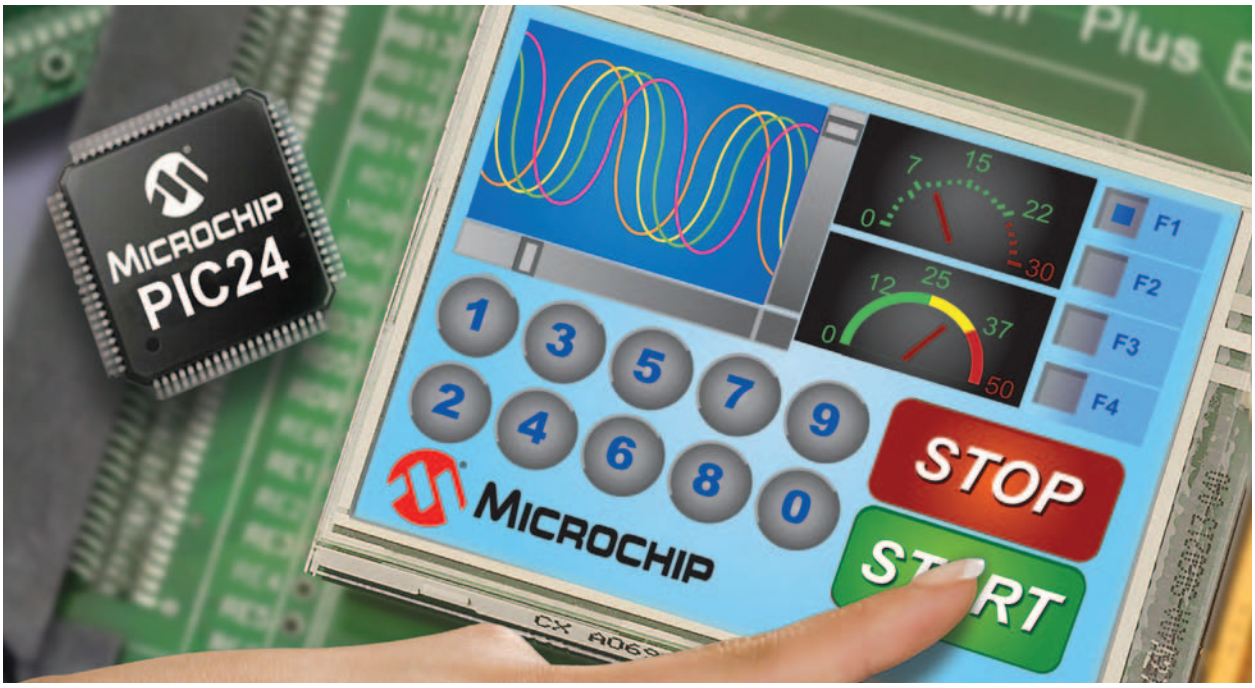
The problem of logic minimization is NP-complete, meaning that there's a combinatorial explosion as the number of inputs, N, grows. This is the nature of the problem, and there is no way out of it. Even so, the Quine-McCusky method remains as one of the most effective logic minimization algorithms. According to Wikipedia, "the runtime of the Quine-McCluskey algorithm grows exponentially with the input size. It can be shown that for a function of n *variables the upper bound on the number of prime implicants is 3^*n/n*." You say that Q-M can't handle values of N larger than 4–5, but for* N = 6*, the number of prime implicants is only about 120. Piece of cake, at that level. Have you considered that your implementation might have a problem?*

Wikipedia goes on, "Functions with a large number of variables have to be minimized with potentially non-optimal heuristic methods..."

If you have truly devised a method that fully optimizes this problem for N *>= 20, your future is assured.*

Send your comments to Richard Nass at rnass@cmp.com or via the Embedded.com Forum, www.embedded.com/tigforums.

Low-Cost QVGA Graphics



Add a graphical user interface to your product with Microchip's 16-bit MCUs, low-cost development tools and software library.

Get started at **www.microchip.com/graphics**:

- View **FREE** web seminars, video demonstrations, application notes and more…
- Download **FREE** graphics library for fast and easy graphics implementation
- Buy **LOW-COST**, **FULL-FEATURED** development tools, including the **Explorer 16 Development Board** (DM240001) and the **Graphics PICtail™ Plus Daughter Board** (AC164127)

FREE LIBRARY

Graphics Features

65,000 colors

320x240 (QVGA) Resolution

Line, Circle, Rectangle, Polygon, Button, Window, Check Box, Slider, Progress Bar, Meter, Image, Animation, Touch Screen, Keypad and More...

Purchase your 16-bit PIC24 microcontrollers and related development tools at...

microchip DIRECT
www.microchipdirect.com

AC164127

MICROCHIP

www.microchip.com/graphics

Microcontrollers • Digital Signal Controllers • Analog • Serial EEPROMs

BY Jack W. Crenshaw

Who needs matrices?

Now that we've completed the vector class, it's time to move on to the matrix (plural: matrices). In the worlds of math, engineering, and physics, it's the matrix that separates the men from the boys, the women from the girls, the heroes from the goats. If you can toss around matrices as you might a Frisbee, you're one of the superheroes. If not, you're on the sidelines, watching. Hint: if you say "matricee" as the singular of matrices, you're not even in the stadium.

I want to show you the C++ code that lets you toss around matrix-vector expressions, but at this point, I think it's much more important for me to give you the motivation.

- Why do I need a matrix?
- What good are they?
- How does matrix algebra help me?

If you can toss around matrices as you might a Frisbee, you're one of the superheroes. If not, you're on the sidelines, watching.

JOHN, MARY, ET AL.

Remember those word problems that used to drive you crazy in high school algebra? Here's an easy one, for old times' sake:

1. Between them, John and Mary have $1.00
2. John has three times as much money as Mary.
3. How much money does each child have?

Now, if you've been thinking outside the box, instead of having your eyes glaze over at the phrase "word problem," you already know the answer. Think three quarters in John's hand, one in Mary's.

Maybe that one was too easy. Try this one:

1. Between them, John, Mary, and Sue have $1.00.
2. Sue has 40 cents more than John.
3. Sue has as much money as John and Mary together.

Not so obvious this time, is it? As you may remember, the trick is to write down equations that capture the information in the problem. To keep it brief, let J, M, and S represent the money each child has. Fact #1 says:

$$J+M+S=100 \quad (1)$$

The next one says:

$$S=J+40 \quad (2)$$

And finally:

$$S=J+M \quad (3)$$

Now we need to solve for the individual values. As you may remember, the trick is to manipulate the equations so that we isolate a single variable, one at a time. For example, Equation 2 gives us a value for S. Let's substitute it into the other two equations, to get:

$$\begin{aligned} J+M+(J+40)&=100 \\ J+40&=J+M \end{aligned} \quad (4)$$

A little simplifying gives:

$$2J+M=60 \quad (5)$$

$$40=M \quad (6)$$

Luckily, the terms in J in the last equation cancelled each other, isolating the value of M. We now know that Mary has 40 cents. Substituting that value into Equation 5 gives:

Jack Crenshaw is a senior software engineer at General Dynamics and the author of Math Toolkit for Real-Time Programming, *from CMP Books. He holds a PhD in physics from Auburn University. E-mail him at jcrens@earthlink.net.*

$$2J+40=60$$
$$2J=60-40=20 \quad (7)$$
$$J=10$$

Now that we know the value of *J*, we can solve Equation 2 for *S*:

$$S=10+40$$
$$S=50 \quad (8)$$

The problem is solved. John has 10 cents, Mary 40, and Sue, 50.

I should probably point out that the sequence of steps I used to solve the problem is not at all the only one, or even the best one.

Way back when, we used to call problems like this one a problem in "simultaneous equations." Simultaneous because, of course, the solution values must satisfy all of the given relationships—in this case Equations 1 through 3. As you remember this kind of problem, you will probably also remember that you must have as many equations as you have unknowns. Give me two equations in three unknowns, and I can't give you an answer. But give me three, and I can be sure to solve the problem (unless one of the equations gives us no new information). If you don't remember anything else about this kind of problem, at least remember this—it's the term "*n* equations in *n* unknowns." And, as you can see, the solution comes by trying to isolate the unknowns, one at a time. As you find the value for one of the unknowns, you now have *n*–1 equations in *n*–1 unknowns, and you can continue to whittle the problem down to size.

Give me two equations in three unknowns, and I can't give you an answer. But give me three, and I can be sure to solve the problem . . .

A LITTLE ORDER, PLEASE

Mathematicians have been solving problems like this for, oh, a few thousand years, give or take. For a long time, they were satisfied to treat each problem as it presented itself, trying operations pretty much at random, until they got the answer. But that approach leaves us very much dependent upon our cleverness in deciding which unknown to eliminate first, and how. A process that depends on the cleverness of the mathematician may help pump up the ego of that mathematician, giving the same satisfaction that he might get from solving, say, a crossword or Sudoku puzzle. But we're not talking about feeding egos and giving satisfaction here. There are real problems to be solved, and we'd rather not depend on heroes to do it.

Over the years, I've learned an important fact about advanced methods of mathematics (it works for engineering and physics, too). It's a lesson that took a long time to learn, and I'm giving it to you free: the whole point of advanced mathematics is not to feed egos, but to make goats look like heroes.

We often do that by devising systematic approaches that would and can be done the same by everyone. In other words, reduce a process that calls for cleverness to one that involves simply turning a figurative crank. That's the case with the kind of problem we've been studying here.

As a step in that direction, it was decided that we would always write down the equations with only terms involving the unknowns on the left of the equals sign, and known values on the right. In that format, our equations look like this:

$$J+M+S=100 \quad (9a)$$
$$S-J=40 \quad (9b)$$
$$J+M-S=0 \quad (9c)$$

With the equations in this form, we can solve them very systematically by adding and subtracting the equations (scaled, if necessary) to/from each other. Adding equal values to both sides of an equation won't unbalance it. Everything will still balance as long as we remember to add both sides of the equations. As I do so for this example, I'm going to retain all three equations so you can see how they change at each step.

According to this turn-the-crank process, we always choose to eliminate the first variable first. For our example problem, let's add Equation 9a to 9b, and subtract it from 9c. The new equations are:

$$J+M+S=100 \quad (10a)$$
$$2S+M=140 \quad (10b)$$
$$-2S=-100 \quad (10c)$$

As you can see, the unknown *J* now appears only in the first equation. Purely by happenstance, we managed to remove *M* from the third equation. But because I'm trying to follow a systematic approach here, I'm going to pretend I didn't notice, and go on. The next step, is to use Equation 10b to isolate *M*. Subtracting this equation from Equation 10a gives:

$$J-S=-40 \quad (11a)$$
$$2S+M=140 \quad (11b)$$
$$-2S=-100 \quad (11c)$$

Now *M* appears only in the second equation. To complete the job, we must eliminate *S* from the first two equations. First subtract ½ of Equation 11c from 11a,

Answer: Both.

Which path would you choose? Would you stick to the safer-looking,well-traveled route? Or would you choose the more rugged path — the one that may very well take you on an adventure?

What if you could do both? With new high-performance, low-power Intel embedded technology you get all of the familiar goodness of the most widely used processing architecture in the world. And now we've added advanced platform technologies that will help you blaze new trails and explore new ground. Like hardware-enabled virtualization and active management technology that allows for remote control of embedded platforms. All without adding more chips or increasing the complexity of your board designs.

With so much efficiency built right in, it's almost like going down two roads at the same time. And that's an adventure on a whole new level. Are you ready for it?

To learn more go to: http://developer.intel.com/design/info/906.htm

Intel Embedded Technology. Igniting Innovation

and add it to 11b. We get:

$$J-S+S=50-40 \quad (12a)$$
$$M=140-100 \quad (12b)$$
$$-2S=-100 \quad (12c)$$

or simply:

$$J=10$$
$$M=40 \quad (13)$$
$$S=50$$

which agrees with our previous, more ad hoc solution.

You may have noticed that the more systematic solution can take more operations than we might take if we were more clever. The price of making goats look like heroes is that the process may sometimes take a little longer. It's a price we tend to gladly pay, to keep us from falling short of hero status.

Over time I've observed that if we're able to attach a name to a problem, we're a long ways along toward solving it.

LINEAR ALGEBRA

Over time I've observed that if we're able to attach a name to a problem, we're a long ways along toward solving it. If a doctor can say "you've got a virus" as opposed to, say, "you've got a hangnail," he knows which remedies are likely to work and which ones won't. A lawyer who says, "This is a case of personal liability" rather than a case of murder, can eliminate a lot of the books in his library, in search of a precedent.

It's the same way with math problems. The equations of the kind we've been looking at are of a special kind. Not only are they simultaneous equations, but they're simultaneous *linear* equations—linear because each of the unknowns appears only to the power one. There are no J^2 terms or $\sqrt{J}$ terms. The science of solving problems involving multiple linear equations in multiple unknowns is called linear algebra. Once we have recognized the word problem as one of linear algebra, we're very much farther than halfway to a solution.

Remember this: you always need as many equations as you have unknowns. If you have, say, three equations but four unknowns, the problem is under-specified. You can't solve the problem entirely. It's not that there is no solution to the problem, but rather that there's an infinity of them. The most you can hope for is to be able to express three of the unknowns in terms of the fourth one.

If, on the other hand, you have four equations in three unknowns, the problem is over-specified. In general, no solutions will satisfy all four equations. The exception is if one of the equations is in fact superfluous—a simple restatement, however disguised, of the other three.

In our example, I had three equations—statements of fact—involving the three unknowns *J*, *M*, and *S*. It should go without saying that a simular problem involving 37 equations in 37 unknowns will take longer to solve, but is no different, in principle. If we apply the same turn-the-crank approach (and if—a big if—we don't make a silly math error), we'll eventually get the answer. As my old major professor used to say, "Conceptually, there's no problem."

Before we leave this example, I want to show you how the problem of simultaneous linear equations morphs into a matrix equation. To begin, we first agree to write down all the possible terms in each equation, with explicit coefficients, including 1 and 0. If a given unknown doesn't appear in the equation, we put it in anyway, with a coefficient of zero. We also keep the unknowns in the same order throughout.

Following this rule, Equations 9a through 9c look like this:

$$\begin{aligned} 1J+1M+1S&=100 \\ -1J+0M+1S&=40 \\ 1J+1M-1S&=0 \end{aligned} \quad (14)$$

Looking at this form, we begin to get the idea that the names of the unknowns doesn't matter. If we had called them *x*, *y*, and *z*, it wouldn't have changed the solution. In fact, the essence of the problem is in the array of coefficients themselves. If I give you that array, you are way far along in finding the solution. In our case, that array is:

$$\mathbf{A}=\begin{bmatrix} 1 & 1 & 1 \\ -1 & 0 & 1 \\ 1 & 1 & -1 \end{bmatrix} \quad (15)$$

At this point, I'm going to assert that this array is, in fact, a matrix. By convention matrices are denoted by uppercase, boldfaced characters, while vectors are lowercase and boldfaced. Without knowing anything about matrix-vector math, let's define the two vectors:

$$\mathbf{x}=\begin{bmatrix} J \\ M \\ S \end{bmatrix};\ \mathbf{u}=\begin{bmatrix} 100 \\ 40 \\ 0 \end{bmatrix} \quad (16)$$

Then I can write the problem in the incredibly compact form:

$$\mathbf{Ax}=\mathbf{u} \quad (17)$$

There is no such thing as a division operator for matrices, but there is the next best thing: an inverse operator. So, conceptually, I can write:

$$\mathbf{x}=\mathbf{A}^{-1}\mathbf{u} \quad (18)$$

How does one compute the inverse of a matrix? We've already done it. The steps we took in Equations 9 through 13 are nothing less than the Gaussian elimination leading to the inverse. This one time, we did it the hard way, only one step removed from the high school way. If you never liked those word problems that ended up as simultaneous linear equations, rejoice: this is the last time you'll ever have to do it. Armed with a matrix inverse operator—all the commercial math aids have them, as will our C++ library—you can go from Equation 17 to 18 in a single step.

Hooray!

As a teaser, I'll just toss out there that, while the matrix division operator doesn't exist, I've appropriated the division operator in the C++ class. Thus, turning Equation 18 into C++ code is as simple as writing:

```
x = u/A;
```

An important point may have slipped past you. As we solved the example problem, it was not at all obvious that the input or constraint vector, **u**, is practically immaterial to the solution. The matrix **A** is purely a matrix of the coefficients of the elements of **x**—in our case, *J*, *M*, and *S*. Therefore we can invert the matrix as a standalone entity, completely independently of **u**. In my statement of the problem, I could have substituted any constant values whatever for the elements of **u**. While the resulting values for **x** would absolutely be different, the inverse of the matrix itself would be exactly the same. This fact gives us a lot of leverage if we're looking at different values of **u** at different times. No matter how many such values there are, we only need to invert **A** once.

Many other problems look a lot easier using matrix algebra, and many others are the same problem, in disguise.

OTHER MATRIX APPLICATIONS

Solving word problems is certainly not at all the only problems we can cast into matrix form. Many other problems look a lot easier using matrix algebra, and many others are the same problem, in disguise. A familiar one is linear regression, which in turn is a subset of polynomial regression, which is itself a subset of optimal

A least-squares fit.

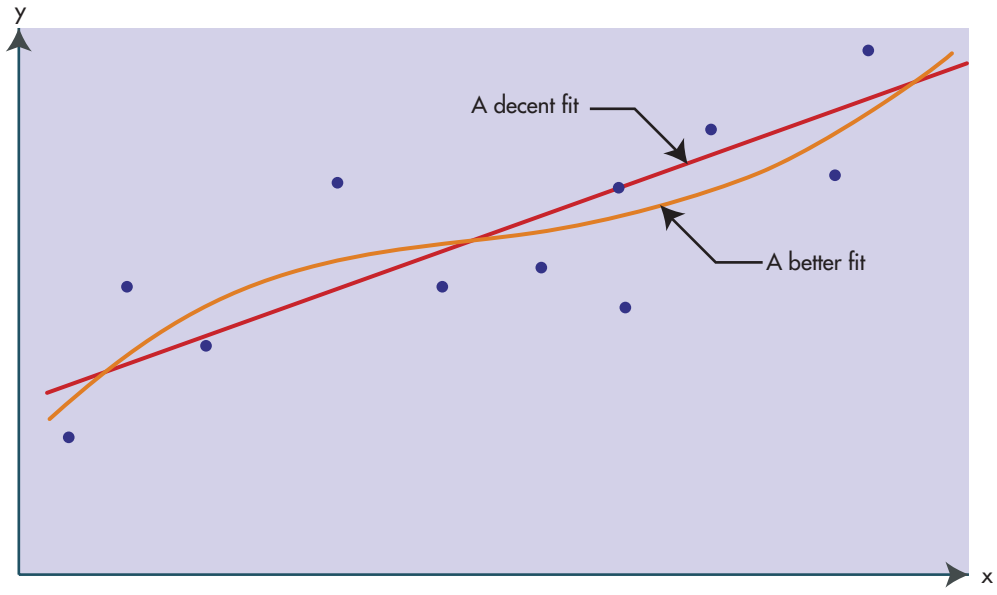


Figure 1

curve fitting. In Figure 1, I've shown a set of data points with a lot of scatter, which we probably got from a sensor beset with lots of noise. Our challenge is to fit a curve through the data.

Suppose I try to draw a straight line through the data. Clearly, it's not going to pass through all the points; chances are, it won't pass through any of them. The equation for a straight line is:

$$y(x)=ax+b \tag{19}$$

where a and b are coefficients to be determined. I can choose lots of different straight lines and *claim* that they're the best fit to the data, but which one is really best?

This is the problem of the least-squares fit. For any of the points, we compare the value of y give by Equation 19, with the measured value. For want of a better criterion, I'm going to claim that the best fit is one that minimizes the sum of the squares of the errors. That's the least-squares fit.

I'm going to be covering this problem in detail in a later column. For now, I only want to say that we can write down the equations for the sum of the squares of the errors. Call it M for historical reasons. To minimize this sum, we end up taking the partial derivatives of M with respect to a and b. The optimal fit occurs when both partials vanish.

For a linear equation, the results are going to look like:

$$\begin{aligned} pa+qb&=r \\ sa+tb&=v \end{aligned} \tag{20}$$

where the constants p, q, etc., depend only on the measurement values, not a or b. Does this look familiar? Yes, indeed, it's our old friend, the simultaneous linear equation. Using the same techniques, we can easily solve for the coefficients of $y(x)$.

When we get into the least-squares problem, you'll see that the size of the array is completely independent of the number of data points. We need at least two to fit a straight line through them, but two or 200,000, it makes no difference. The matrix will be different, but we're still only solving for two unknowns.

When we get into the least-squares problem, . . . the size of the array is completely independent of the number of data points.

What's more, if we like, we can assume a higher-order polyomial. The curve in Figure 1 looks like it might be a cubic, so will have four coefficients. The number of coefficients makes little difference. The only difference between them is the size (order) of the matrix. And even when we have many, many data points, the order is modest.

THINGS TO COME

Simultaneous linear equations and the least-square fit. Seemingly unrelated problems, but they both fall before the might of the matrix inverse. But these two disciplines are not by any means the limit to the value of matrix algebra. Other areas include:

- Coordinate transformations.
- Rotational dynamics.
- Multivariate control theory.
- Optimal control theory, including the Kalman filter.

As we go through the steps to define a matrix class, I'll be showing you, as I did for the vector class, practical applications. I think you'll enjoy the trip. ■

To NAND or NOR. That is the question. Different applications and functions should be handled with different types of flash memory.

Employ

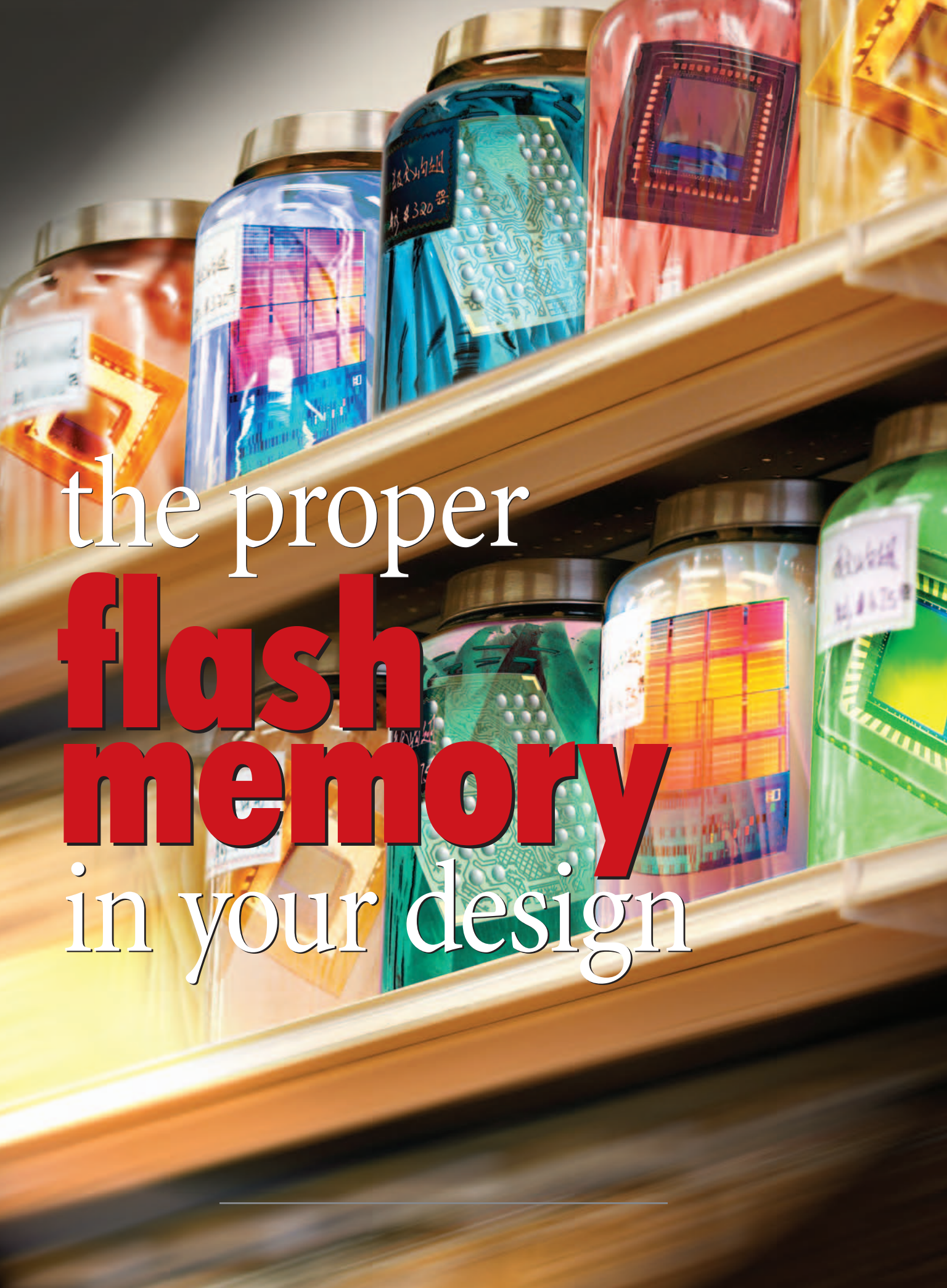
BY VIJAY K. DEVADIGA

In their search for the perfect "universal memory," designers of embedded systems are like characters from the play *Waiting for Godot*. They're waiting for an off-stage character, "Godot," to come on stage, meanwhile speculating which of the current or new characters appearing on stage are Godot.

From it's inception in the 1970s, the embedded systems industry as we know it as always been waiting for an off-stage semiconductor-based character named "universal memory" to come along and replace the memory hierarchy inherited from mainframes, minicomputers, and desktop computers: nonvolatile hard disk drives for long term mass storage and backup, dynamic RAM for local fast memory access, and SRAM and ROM for extremely fast access and code storage.

This desire has been heightened and intensified as computing as become more embedded, mobile, and portable and candidates for the role of "universal memory" have come on stage. Some—such as EEPROMS, EPROMs, UV-EPROMs, ferroelectric RAMs, and various pseudo-RAM combinations—have been turned down. Others such as magnetic RAM are being considered, but are in doubt for a number of economic and technical reasons.

However, some of the current characters on the stage—particularly the various flavors of NAND and NOR flash EPROMs—are being touted by their vendors as that offstage character "universal memory," or at least a relative and "close personal friend" including OneNAND, OrNAND, iNAND, GBNAND, moviNAND, ManagedNAND and NANDrive. Giving the diversity of actors trying out for the role, choos-



the proper flash memory in your design

ing the right memory subsystem is much more complicated now, especially if you're adding more multimedia functions to mobile and embedded systems while shrinking physical size and reducing overall system cost. Not only could the code and data storage needs have increased in these systems, but you've got to do it all more reliably with less of everything.

Flash memory is the most practical solution, but knowing which type of flash fits best in a system is the key. Is NAND, NOR, managed NAND, or some hybrid the best choice?

The use of NAND flash, an inexpensive and high-density nonvolatile memory requiring defect management, to satisfy these growing code and data storage needs makes the memory subsystem even more complex. Add to that the need to support different memory types, interfaces, vendors, and vendor-specific features, and the memory subsystem even more complex.

A completely managed memory subsystem solution can be designed that uses an industry-standard RAM (PSRAM or SDR/DDR SDRAM) interface. This managed memory subsystem would provide seamless integration with the host chipset/processor and eliminate the need for the host system to manage the complexity and deficiency of built-in memory devices.

This system has a host chip set that interfaces with standalone NAND flash.

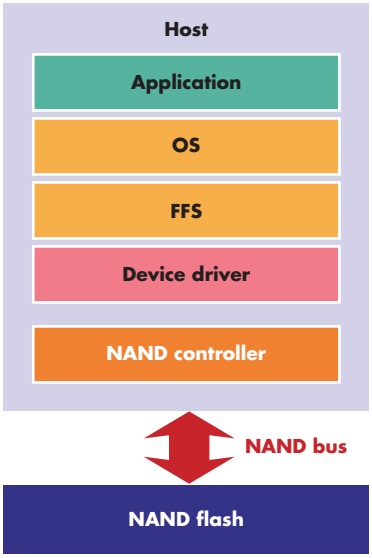


Figure 1

Unlike NAND flash, NOR flash is one of the oldest and the most widely used memory types in current embedded systems. It's used for both code and data storage. Its main advantage is that the code is executed directly (execute-in-place) from the NOR flash memory. Also, NOR flash can directly interface with the host processor, which enables easy design-in and fast time-to-market.

With the increased deployment of multimedia functions in embedded systems, the need for code and data storage is also increasing. For these applications, using higher density NOR flash for code and data storage becomes more expensive when compared with alternative solutions like NAND flash. In addition, the highest density NOR flash available today is 1 Gbit. Moreover, multimedia data storage requires both high read and write performance. As a result, system designers have turned to NAND flash for storing multimedia files as well as application code in many embedded applications, including high-end cell phones.

NAND FOR CODE & DATA STORAGE

NAND flash is well suited for applications that require large code storage (such as operating system and applications) and large data storage because NAND flash is inexpensive and is also available in high densities (up to 16 Gbits in one die). Unlike NOR, NAND flash doesn't support execute in place (XIP) or random access. As a result, some systems that use NAND flash need a low-density NOR flash just for system boot-up and BIOS code execution. In other systems, a NAND flash controller, or an embedded boot ROM in the host processor, can provide the boot function. After system boot-up, the NAND-based systems use either code shadowing or demand paging for code execution. In the case of code shadowing, the entire operating system and applications are copied from the NAND flash into the system RAM, and in demand paging, a portion of the operating system and applications are

Family of High-Efficiency, Full-Featured PowerWise® Synchronous Buck Regulators

national.com/switcher

Integrated Features for Optimized Power Supply Designs

LM20xxx Family Features

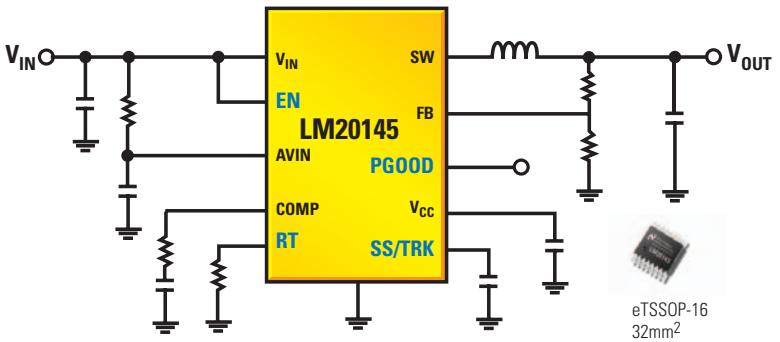
- External soft-start
- Tracking
- Precision enable
- Power good
- Pre-biased start-up
- Enhanced system reliability
  - High accuracy current limit
  - Over-voltage protection, under voltage lockout, and over-current protection
- Available in eTSSOP-16 packaging

Feature Options

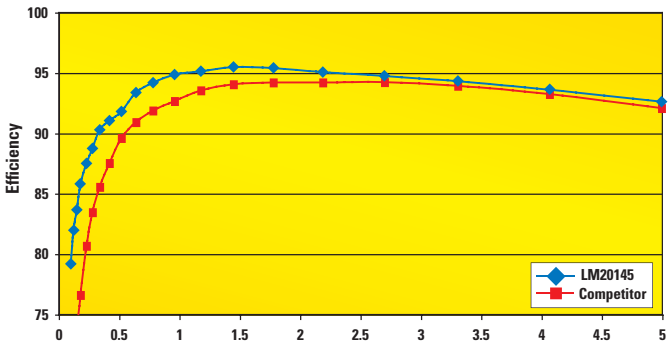
- Fixed and adjustable switching frequency
- Clock synchronization in
- Clock synchronization out

Ideal for powering FPGAs, DSPs, and microprocessors in servers, networking equipment, optical networks, and industrial power supplies

Highest Power Density 5A Regulators



Efficiency vs Output Current (V_{IN} = 5.0V, V_{OUT} = 3.3V, fSW = 500 kHz)



Product Number	V_{IN} (V)	I_{OUT}	SYNC IN	Frequency Adjust	SYNC OUT	Frequency
LM20123	2.95 to 5.5	3				1.5 MHz
LM20133	2.95 to 5.5	3	✔			Sync
LM20143	2.95 to 5.5	3		✔		500 kHz - 1.5 MHz
LM20124	2.95 to 5.5	4				1 MHz
LM20134	2.95 to 5.5	4	✔			Sync
LM20144	2.95 to 5.5	4		✔		500 kHz - 1.5 MHz
LM20154	2.95 to 5.5	4			✔	1 MHz
LM20125	2.95 to 5.5	5				500 kHz
LM20145	2.95 to 5.5	5		✔		250 kHz - 750 kHz
LM20242	4.5 to 36	2		✔		250 kHz - 750 kHz

For FREE samples, datasheets and more, visit:

national.com/switcher

Or call: 1-800-272-9959

National Semiconductor
The Sight & Sound of Information

copied into the system RAM as needed, where such code is then executed.

Though NAND flash is inexpensive and available in higher densities than NOR, NAND is less reliable and requires defect management, including error detection and correction, and wear-leveling to make it usable for many applications. These NAND flash management functions require complicated hardware and software. Figure 1 shows a system where the host chip set interfaces with standalone NAND flash. In this system, the defect management functions must be performed by the host chip set. Running such flash management functions on the host requires some software development and it also uses some of the host's CPU and memory resources, leading to a reduction in overall system performance.

As NAND flash vendors are moving to smaller process geometries, the ECC (error correction code) requirement for single-level cell (SLC) NAND flash has increased from 1 to 4 bits per 512-byte sector, and for multi-level cell (MLC) NAND flash, it's increased from 4 to 8 bits per 512-byte sector. The page size has increased from 512 to 4,096 bytes. The endurance for some of the smaller geometry SLC NAND flash has changed from 100,000 to 50,000 cycles, and for MLC NAND flash it has changed from 10,000 to 5,000 cycles (and 3,000 cycles in some cases). To reduce the number of discrete components in the system, many chip set vendors have started integrating a NAND flash controller in their chip set, which can directly interface with standalone NAND flash. However, because of the long design cycle of a chip set, it's difficult for the chip set vendors to track the ever changing NAND flash technologies. Therefore, the functionality of an embedded NAND flash controller in the chip set will always lag the NAND flash technologies.

A few solutions are similar to standard NAND flash, but offer improved performance and functionality. For example, OneNAND is a variation of NAND flash that combines RAM and standalone SLC NAND flash in one device to provide boot function and faster read access. OneNAND requires 1-bit ECC for each 512-byte sector and requires NAND flash management functions implemented either in the host chip set or in a separate standalone controller.

On the other hand, OrNAND combines MirrorBit NOR with a NAND flash interface, which offers faster write time than its predecessor NOR devices. OrNAND also requires 1-bit ECC implemented either in the host chip set or in a separate standalone controller to ensure reliable system boot-up. Moreover, the maximum density currently supported by OrNAND is only 1 Gbit, which is less than the maximum density of currently available NAND flash.

MANAGED NAND FOR DATA

Due to the limitations of the embedded NAND controllers, many system design-

Managed NAND memory, used mainly for data storage, reduce system complexity by managing the built-in NAND flash with a NAND controller and flash file system integrated into the same device.

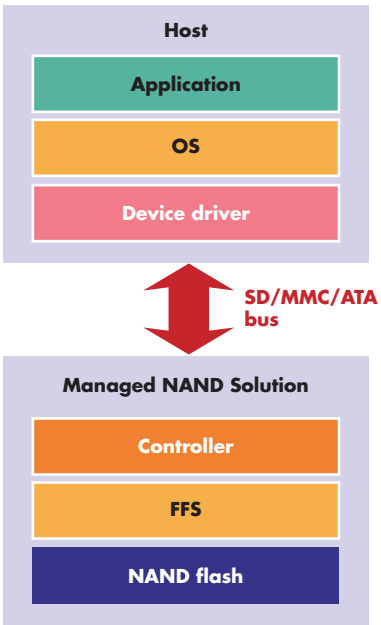


Figure 2

ers are looking at managed NAND solutions. Several vendors have come up with managed NAND memory products that alleviate the complexities of a conventional memory subsystem in embedded applications. These managed NAND memory products, including iNAND, GBNAND, moviNAND, Managed NAND, and NANDrive, are used mainly for data storage. They reduce system complexity by effectively managing the built-in NAND flash with a NAND controller and flash file system (FFS) integrated into the same device as shown in Figure 2. These products use a standard interface such as Secure Digital (SD), MultiMediaCard (MMC), or Advanced Technology Attachment (ATA) for easier integration. For example, iNAND and GBNAND offer the SD interface, moviNAND and Managed NAND offer the MMC interface, and NANDrive offers the ATA interface. These products don't offer XIP access and as a result, systems that use them may need a NOR flash for the boot function.

Using a managed NAND device eliminates the need for a complex NAND management function on the host. As a result, the chip-set vendors needn't worry about keeping up with the evolving NAND technologies, thus enabling vendors to focus more on their core competencies.

MANAGED NAND HYBRID BOOTS

Because managed NAND flash doesn't provide boot capability, system designers still must use a higher cost NOR flash device for boot-up. However, hybrid products, including mDOC H3, are now available. These hybrids use RAM and managed NAND in a one device, as depicted in Figure 3, to simplify the conventional memory subsystem.

Hybrid products solve the boot issue associated with the managed NAND. They can boot directly from the NAND flash, eliminating the need for a higher cost boot NOR flash device, which may reduce the overall system cost. Managed NAND hybrids also help to reduce component count and save board space, which makes them suitable for space-sensitive applications like cell phones. These solutions are available in higher densities because they use NAND flash for nonvolatile storage.

On the down side, NAND hybrids have a longer boot time because they must copy the boot code from the NAND into the boot RAM after power-on. Also, NAND hybrids are complex, difficult to integrate, and require an advanced operating system that supports demand paging on the host.

mDOC H3 uses a NOR-type bus to interface with the host processor and offers faster read performance than NAND flash and faster write performance than NOR. Due to the faster write performance, these devices are suited for storing multimedia.

Using a managed NAND or even a managed NAND hybrid with boot capability doesn't drastically reduce the memory subsystem's overall complexity. The system designers must still manage the complexity associated with different memory types, interfaces, vendors, vendor-specific features, and so forth. These types of memory subsystems require many components, more pins, and complex hardware and software development, resulting in increased system cost, board space, development time, and power consumption. At the same time, they increase the complexity of the external memory controller in the host processor. Therefore, these hybrids are less than complete. Today's systems need an easy-to-use, single-standard bus, completely managed memory subsystem for code and data storage and system RAM, all in one device.

What system designers need is a complete memory subsystem that offers hundreds of megabytes of XIP code storage and also satisfies the growing data-storage needs of today's multimedia applications. The solution should blend the key benefits of NOR (fast read), NAND (lower cost and higher density), and RAM (simple bus operation). This solution also must be easy to use, easy to design in, and completely managed. Thus, it requires minimal or no additional hardware or software development, offers a standard interface for seamless integration with the host chip set/processor without any glue logic, and makes memory subsystem access as simple and easy as an SRAM.

The built-in controller in this memory subsystem solution must implement ECC, bad block management, and wear-leveling to manage the defects in the built-in NAND flash. The controller must also manage the complexity and deficiency of all the built-in memories (NOR, NAND, and RAM) to off-load the host system from handling these complicated management functions.

Furthermore, this solution must address the total system cost by reducing the material, development, and manufacturing costs. In addition, it must address the time-to-market needs of system designers working on mobile and consumer electronic devices by offering a solution that's easy to integrate and is scaleable for future products.

NEXT GENERATION SUBSYSTEM

Managed memory subsystem products are available today that offer all the features and benefits mentioned. For example, one such configuration shown in

Some hybrid solutions use RAM and managed NAND in a one device.

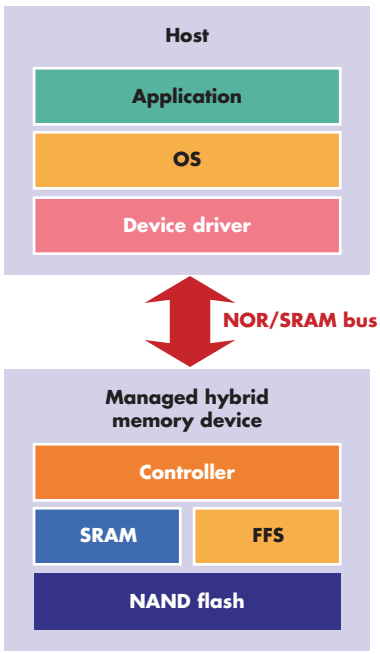


Figure 3

All-in-OneMemory is an example of a managed memory subsystem. It consists of a memory controller with built-in boot NOR flash, NAND flash, and RAM, in one package (a). The controller handles on-demand paging and other memory management functions (b).

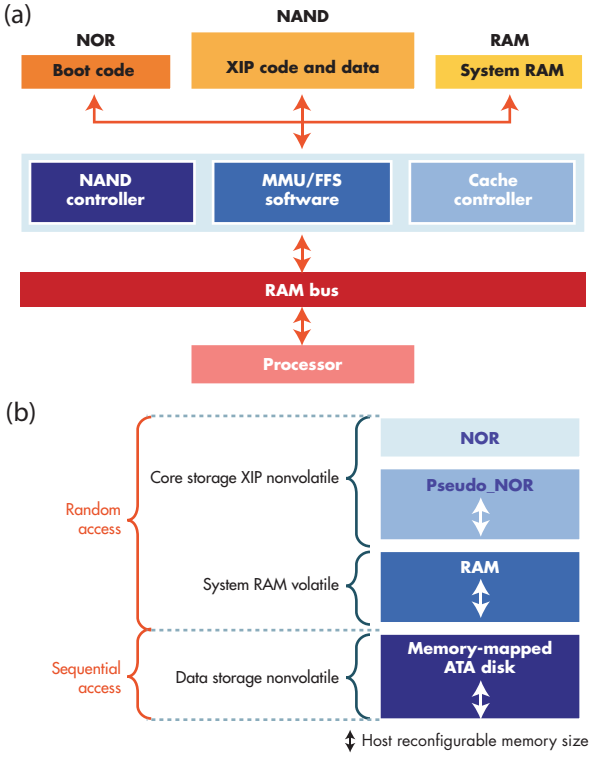


Figure 4

Figure 4 consists of a memory controller with built-in boot NOR flash, NAND flash and RAM, all in one package. Using the RAM cache in front of the NAND flash, the controller handles on-demand paging and other memory management functions. In addition, the RAM cache renders the memory subsystem addressable linearly, and as simple as an SRAM.

The RAM block is divided into two host-accessible, user-configured partitions: a cache partition for Pseudo-NOR (PNOR) and a system RAM partition for the host. The NAND block is used as nonvolatile storage for the PNOR area and the memory-mapped ATA NAND disk area. The configurable PNOR block emulates the NOR function by using RAM cache and NAND flash. Because NAND is used as the main nonvolatile storage media, this solution provides large XIP code storage that can effectively replace the traditional higher cost, high-density NOR flash. By using the industry standard ATA data-storage protocol on the standard RAM (PSRAM or SDR/DDR SDRAM) bus, this solution provides large ATA-like data storage for the growing multimedia applications. In addition, the RAM cache in the PNOR block also helps to extend the endurance and reliability of the code and data storage area by minimizing direct read/write access to the NAND flash.

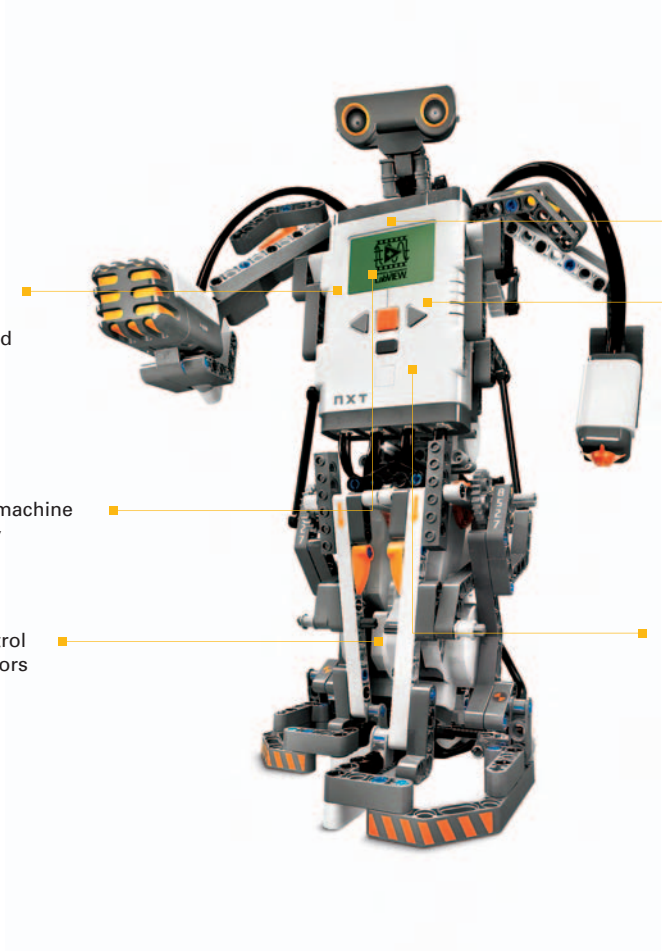
Because it's offered in a small-footprint package, this managed memory subsystem may simplify the host interface, reduce system complexity, shorten design time, reduce overall system cost, and improve quality and reliability. Other benefits include the user-configurable XIP PNOR area; robust hardware error detection and correction for MLC and SLC NAND; and scalability to higher densities. With no complicated software and hardware development required, this type of managed memory subsystem could be the Godot we're all waiting for. ■

Vijay Devadiga is currently a senior staff product marketing engineer at Silicon Storage Technology Inc. in the NAND and Smartcard Module Business. In this role, he is responsible for product marketing of the company's All-in-OneMemory product line. He can be reached at vdevadiga@sst.com.

NEW VERSION AVAILABLE

NI LabVIEW.

Limited Only by Your Imagination.



Real-Time and Embedded | Signal Processing | High-Performance Test | Industrial Control

PRODUCT PLATFORM

LabVIEW Real-Time Module

LabVIEW FPGA Module

LabVIEW Microprocessor SDK

NI CompactRIO Embedded Hardware Platform

When the LEGO Group needed parallel programming and motor control tools intuitive enough for children, it selected graphical software powered by NI LabVIEW. With LabVIEW graphical system design, domain experts can quickly develop complex, embedded real-time systems with FPGAs, DSPs, and microprocessors.

>> Expand your imagination with technical resources at **ni.com/imagine** **866 337 5041**

NATIONAL INSTRUMENTS

feature

Combining a programmable solution with an industry-standard processor core can save time, money, and real estate.

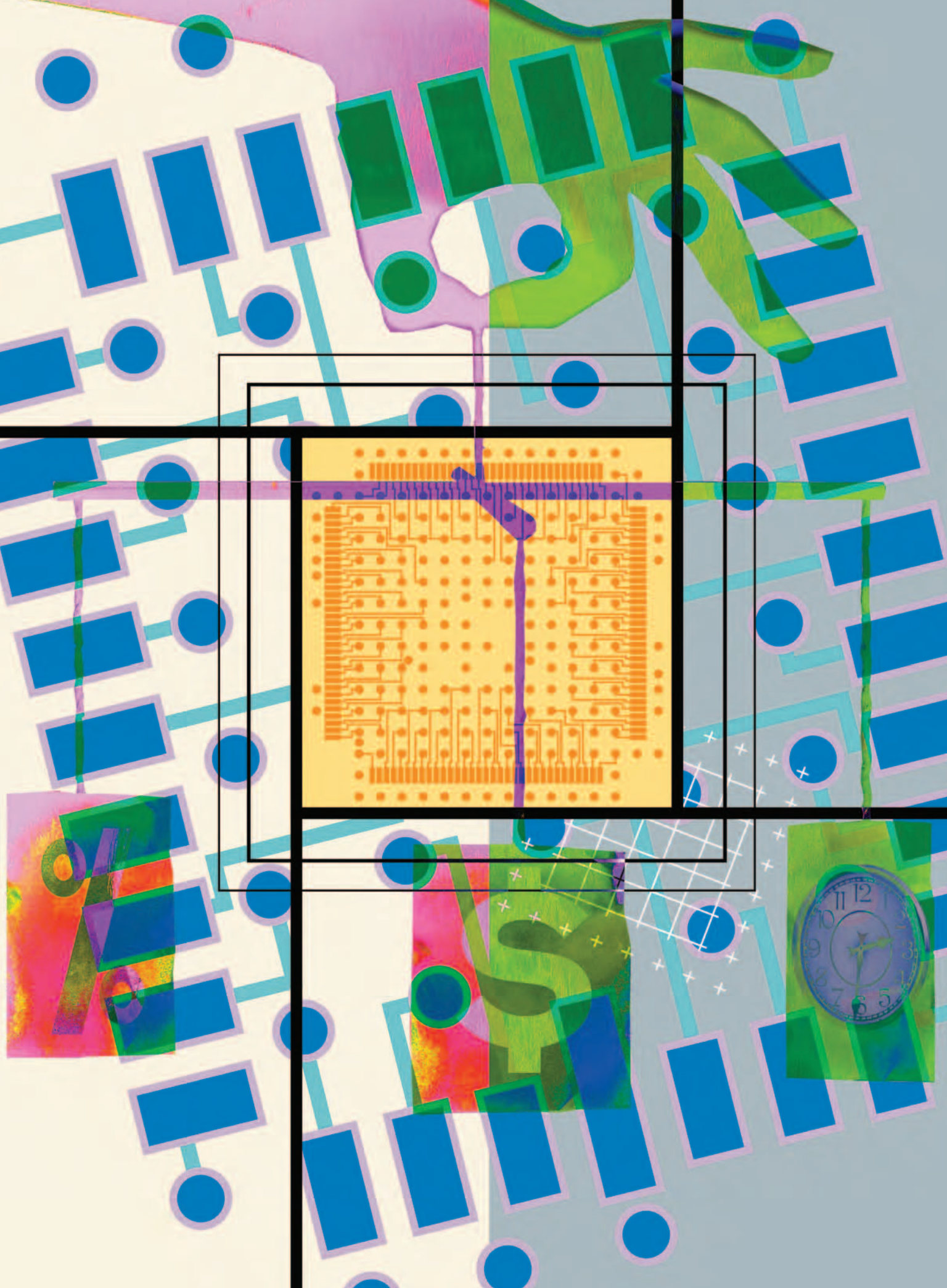
Lower the cost of intelligent power control with FPGAs

BY WENDY LOCKHART

The availability of ARM's Cortex-M1 32-bit processor and single-chip, mixed-signal FPGAs make possible the development of intelligent power control that could dramatically reduce part count, board space, and system cost while increasing reliability, flexibility, and system availability. Methods exist for rapid prototyping and implementation of the hardware and software for server-based intelligent power; these methods can also be applied to a range of energy-management applications. System considerations include development resources; tools available for development with Cortex-M1 and mixed-signal programmable system chips (PSCs); and availability of power control boards. Lowering the cost of intelligent power control requires an understanding of FPGA implementation strategies, as well as a basic understanding of designing with ARM's Cortex-M1 microprocessor and FPGA implementation tools.

As the number of functions in a system grows, power to the system is no longer an afterthought. With time-to-market pressures and the need to support more features, designers must select peripherals from a standard set of components to meet their power and cost budgets. As a result, this may mean mixing an LCD with a 2.5-V supply and a keypad with a 1.8-V supply. With many devices, the core and I/O voltage within the device will also vary. So, within a single product, the power supply may need to generate multiple voltages and possibly different sequences of the same voltage.

In portable applications, the restrictions can be even greater, requiring power management to extend battery life. Designs can therefore become quite complex, featuring multiple supplies, controlled ramp rates, power sequencing, and complex supply manage-



ment where supplies are turned on and off as needed.

Intelligent power control has existed in some high-end systems as a custom implementation or more recently with standards, such as ATCA and MicroTCA. With the market now demanding smaller, portable versions of many applications, the power control must be scaled down as well, creating a need for intelligent power control that's low cost, small form factor, and low power.

Intelligent power control involves the following basic aspects:

- Generating all the required system voltages.
- Sequencing each device's power up and down to maintain system integrity and prevent issues such as latch-up, inrush current, or I/O contention.
- Supporting the ability to switch off certain devices when not needed and power them back up for seamless operation.
- Maintaining minimum functionality in standby, with the ability to wake at certain intervals or on-demand.

If the PSC can perform wake up from a watchdog timer or external trigger, the entire system can be shut down except for one device.

Implementing these functions in an application-specific standard product (ASSP) would require a standard power profile. The use of a programmable device, however, allows for adaptation to multiple system requirements. Programmable devices also allow for future system upgrades and fine tuning. A number of programmable power modules are available, but they still need to be combined with some form of brain to tell them when to turn each supply on and off.

As shown in Figure 1, mixed-signal PSCs allow for the combination of programmable power generation with real-time analysis and control. Combining functions for power sequencing, monitoring, and control in a single chip results in significant cost savings in terms of materials list, board space, and build costs. In addition, a single device consumes less power and can manage the power supplies for the rest of the system. And, if the PSC can perform wake up from a watchdog timer or external trigger, the entire system can be shut down except for one device. Some PSCs can also manage smart battery charging, which is another intelligent way to enhance power control in the system.

Early adoption of PSCs in intelligent power control applications has been facilitated by the use of reference designs, some in the area of ATCA and others in the associated intellectual property (IP), such as Intelligent Platform Management Interface (IPMI).

Example of a mixed-signal PSC architecture.

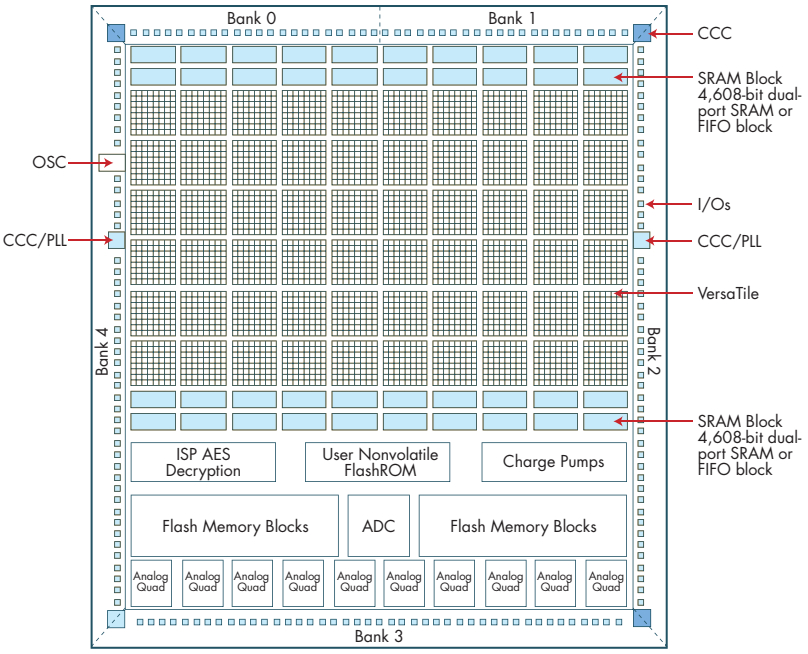


Figure 1

Go wireless with AVR® Z-Link®

The Z-Link Solution

Atmel®'s AVR Z-Link product line provides the best IEEE 802.15.4 compliant and ZigBee™ certified solution available. Based on the AVR RF family of radio transceivers, AVR microcontrollers, free software, reference designs and development kits, the solution enables deployment of large wireless networks with very low energy consumption.

RF radio transceivers provide the widest link budget available with -101 dBm receive sensitivity and 3 dBm transmit power. Combined with picoPower™ technology used by the AVR microcontrollers, the chipset achieves very low power consumption.

The AVR family of 8-bit RISC microcontrollers which features up to 256 KB Flash memory and rich peripherals, can easily handle both wireless communication and your main application.

AVR Flash microcontrollers
2.4 GHz RF transceivers

Hardware

All free!
Transceiver Access Toolbox
Software MAC
ZigBee Stack

Software

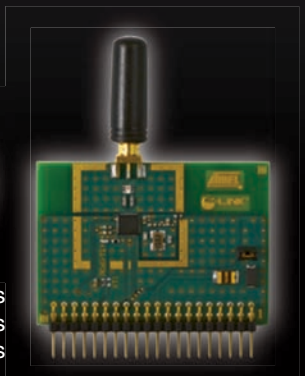
Z-Link

Partners

Hardware design
Stacks development
Wireless gateways
System software

Tools

Evaluation kits
Demo kits
One toolchain for all AVRs



Get more at: **www.atmel.com/AVRman**

ATMEL®

This platform-based approach lets designers start from a known system and customize it, or use a fixed standard and work it into their system. Some system management development kits also provide examples for a more custom approach to power control. This still requires a clear understanding of the PSC as well as a custom development environment.

Because the PSC's core includes standard flash-based FPGA gates, the Cortex-M1 core can be used as part of the design. This provides significant benefits to an existing ARM designer who must integrate intelligent power control without having to learn a full suite of analog tools. Cortex-M1 is fully backwards compatible with ARM's Thumb architecture, so designers can port existing application code to the PSC.

Although implementing a Cortex-M1 design on a PSC might be a new challenge to some engineers, certain tools and techniques can make this a relatively painless design cycle.

Although implementing a Cortex-M1 design on a PSC might be a new challenge to some engineers, certain tools and techniques can make this a relatively painless design cycle. Often the easy approach from a time-to-market point of view is the expensive approach from a cost point of view. But when the design tools are created by a silicon supplier, the motivation is to help the designer get to silicon as fast as possible.

IMPLEMENTING INTELLIGENT POWER CONTROL

Standard FPGA design can take several different forms:

- Full HDL code or schematic starting from scratch.
- Combination of existing versions of design plus some new code and features.
- Combination of purchased or custom IP, existing design, and new code.

FPGA development flow with soft processor.

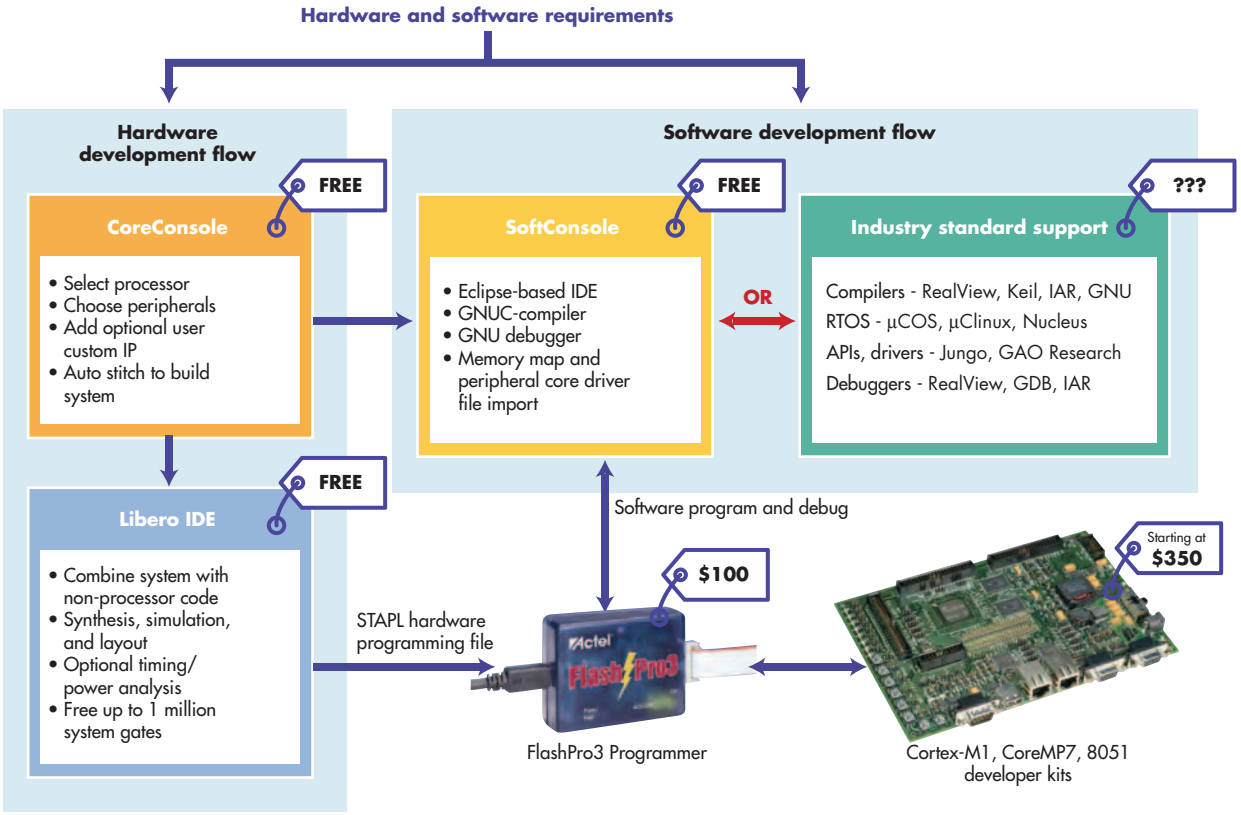


Figure 2

KEIL™
An ARM® Company

Microcontroller Development Tools

Only 4 Steps...

...are required to generate efficient, reliable applications with the µVision IDE and development tools from Keil.

Step 1. *Select Microcontroller and Specify Target Hardware*

Use the Keil Device Database (**www.keil.com/dd**) to find the optimum microcontroller for your application.

In µVision, select the microcontroller to pre-configure tools and obtain CPU startup code.

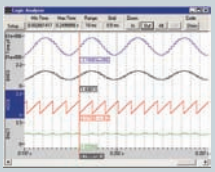
Step 2. *Configure the Device and Create Application Code*

The µVision Configuration Wizard helps you tailor startup code to match your target hardware and application requirements.

Extensive program examples and project templates help you jump-start your designs.

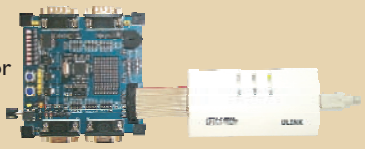
Step 3. *Verify Program Execution with Device Simulation*

High-speed simulation enables testing before hardware is available and helps you with features like instruction trace, code coverage, and logic analysis.

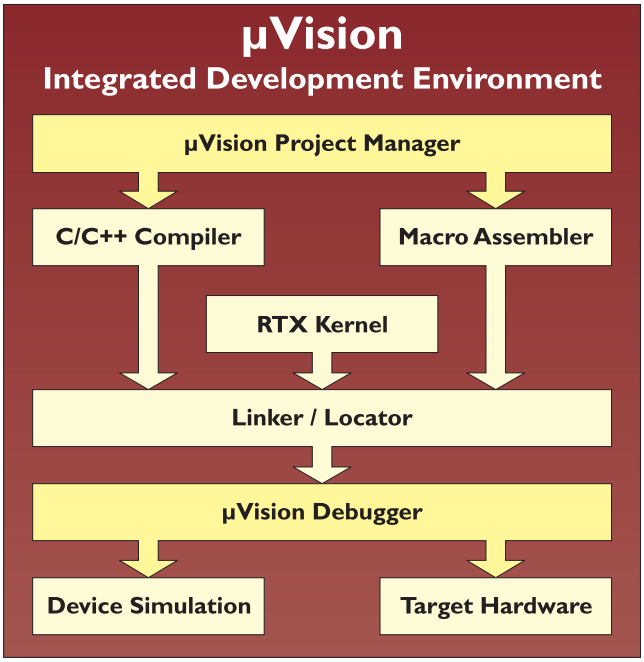


Step 4. *Download to Flash and Test Application*

Once your application runs in simulation, use the Keil ULINK USB-JTAG Adapter for Flash programming and final application testing.



Keil Microcontroller Development Tools help you create embedded applications quickly and accurately. Keil tools are easy to learn and use, yet powerful enough for the most demanding microcontroller projects.



Components of Keil Microcontroller Development Kits

Keil makes C compilers, macro assemblers, real-time kernels, debuggers, simulators, evaluation boards, and emulators.

Over 1,200 MCU devices are supported for:

- **8-bit** - 8051 and extended 8051 variants
- **16-bit** - C166, XC166, and ST10
- **32-bit** - ARM7, ARM9, and Cortex-M3

Download an evaluation version from
www.keil.com/demo

800-348-8051 **www.keil.com**

The same styles apply to both mixed-signal FPGAs and soft ARM processors on FPGAs.

The hardware and software implementation steps are shown in Figure 2. To simplify terms, the hardware implementation involves designing the processor and programming it to the FPGA. The software implementation involves developing code for the processor on the FPGA to complete the application.

First, the designer identifies available industry-standard processors (8051 or ARM, for example) for the design and then looks through offerings from each supplier to find the best list of peripherals for the design. If the perfect list of peripherals can't be found, then a CPLD or companion FPGA can be added to the board. When implementing a soft processor on an FPGA, the supplier should provide a list of available peripherals.

Ideally, the user selects the processor from a list. Then, the bus type, which is often dictated by the processor, is chosen. Once the bus is selected, the user simply drags and drops peripherals into the design. Peripherals may include interfaces such as serial peripheral interface, universal serial bus, timers, or UARTs. Unlike an ASSP processor, users can add any combination of peripherals until they run out of peripheral ports. The tool should provide the ability to auto-connect peripherals where possible and allow users to configure all other parameters and connections through a simple GUI.

Having designed the processor, users then generate and export to the FPGA layout tool. The processor may only be part of the FPGA design. The design may include additional glue logic or interfaces that aren't controlled by the processor. Using a simple block design tool, users can drop in the processor and any additional blocks to create a single top-level design for synthesis.

The design then follows a standard FPGA flow: synthesis, place and route, timing analysis, and so forth. Ideally, the Cortex-M1 processor should be delivered with a custom layout from the silicon vendor, indicating that timing analysis and optimization have been done. This ensures maximum performance in a minimum footprint. The layout should be customized to the device you plan to use. From here, the custom processor can be programmed to the PSC.

What you really want is to convince yourself that both the technology and your design have the ability and functional accuracy to eliminate the need to build custom prototypes—or at least skip the first round of prototyping.

BOARD OPTIONS

The choice of development environment is important. Buying a "demo kit" usually means you can get the design or board to do what the supplier intended. What you really want is to convince yourself that both the technology and your design have the ability and functional accuracy to eliminate the need to build custom prototypes—or at least skip the first round of prototyping. Look for a board that has the interfaces built-in, and has enough I/O available with connectors to work with your regular platform or peripheral board. In addition, key components, such as crystals that can switch out, increase the board's usability as a true prototyping environment. You should also be able to program and debug on the development platform.

Software implementation is similar to a standalone processor. The tool that builds the processor should export the appropriate files required by the compiler to create the application. Existing functions can be exported and enable the ability to work from a library of functions to control the peripherals. Custom code can then be added on top of this. For many processors, including Cortex-M1, the GNU environment offers free tools including compilers and debuggers. However, extra money does get you more mileage. Generally, the more expensive compilers will provide either better performance or less code space. Still, free compilers will get you a functionally accurate design and a long way through prototyping. If you're working with a lot of library functions, the functions should already be optimized, eliminating the need for expensive compilers.

Debug is the next phase of design. With a soft processor, the ability to debug both the hardware and software is important. Again, a preoptimized version of the processor helps, as processor debug won't be necessary. Some tools are designed to coexist, such as ModelSim, which runs the hardware, and GNU, which runs the processor application. It's best to get these tools from your silicon supplier, because the PSC is unique and other debug tools won't be set up to handle the architecture. Also, with a standard ARM processor, existing ARM debug tools support debug of the processor application, but not any additional FPGA logic.

All of the methods and strategies described apply to intelligent power control, whether the application is a simple handheld portable device or for a multiserver rack system. The desire to conserve energy and cost will continue to drive electronics in this direction, while also driving designers to meet these challenges for years to come. ■

Wendy Lockhart is a principal engineer at Actel. She holds a master's degree in electronics from the University of Edinburgh, Scotland. Lockhart can be reached at wendy.lockhart@actel.com.

EMBEDDED

Unlock your future

Enter the New Era of Configurable Embedded Processing

Adapt to changing algorithms, protocols and interfaces, by creating your next embedded design on the world s most flexible system platform. With the latest processing breakthroughs at your fingertips, you can readily meet the demands of applications in automotive, industrial, medical, communications, or defense markets.

Architect your embedded vision

- *Choose MicroBlaze,™ the only 32-bit soft processor with a configurable MMU, or the industry-standard 32-bit PowerPC® architecture*
- *Select the exact mix of peripherals that meet your I/O needs, and stitch them together with the new optimized CoreConnect™ PLB bus*

Build, program, debug . . . *your* way

- *Port the OS of your choice including Linux 2.6 for PowerPC or MicroBlaze*
- *Reduce hardware/software debug time using Eclipse-based IDEs together with integrated ChipScope™ analyzer*

Eliminate risk & reduce cost

- *No worry of processor obsolescence with Xilinx Embedded Processing technology and a range of programmable devices*
- *Reconfigure your design even after deployment, reducing support cost and increasing product life*

Order your complete development kit today, and unlock the future of embedded design.

Get the Complete Embedded Solution

www.xilinx.com/processor

XILINX®

www.xilinx.com/processor

At the Heart of Innovation

High-end concepts typically aren't applied to an 8-bit MCU. But don't rule them out just yet.

The basics of embedded multitasking on a PIC

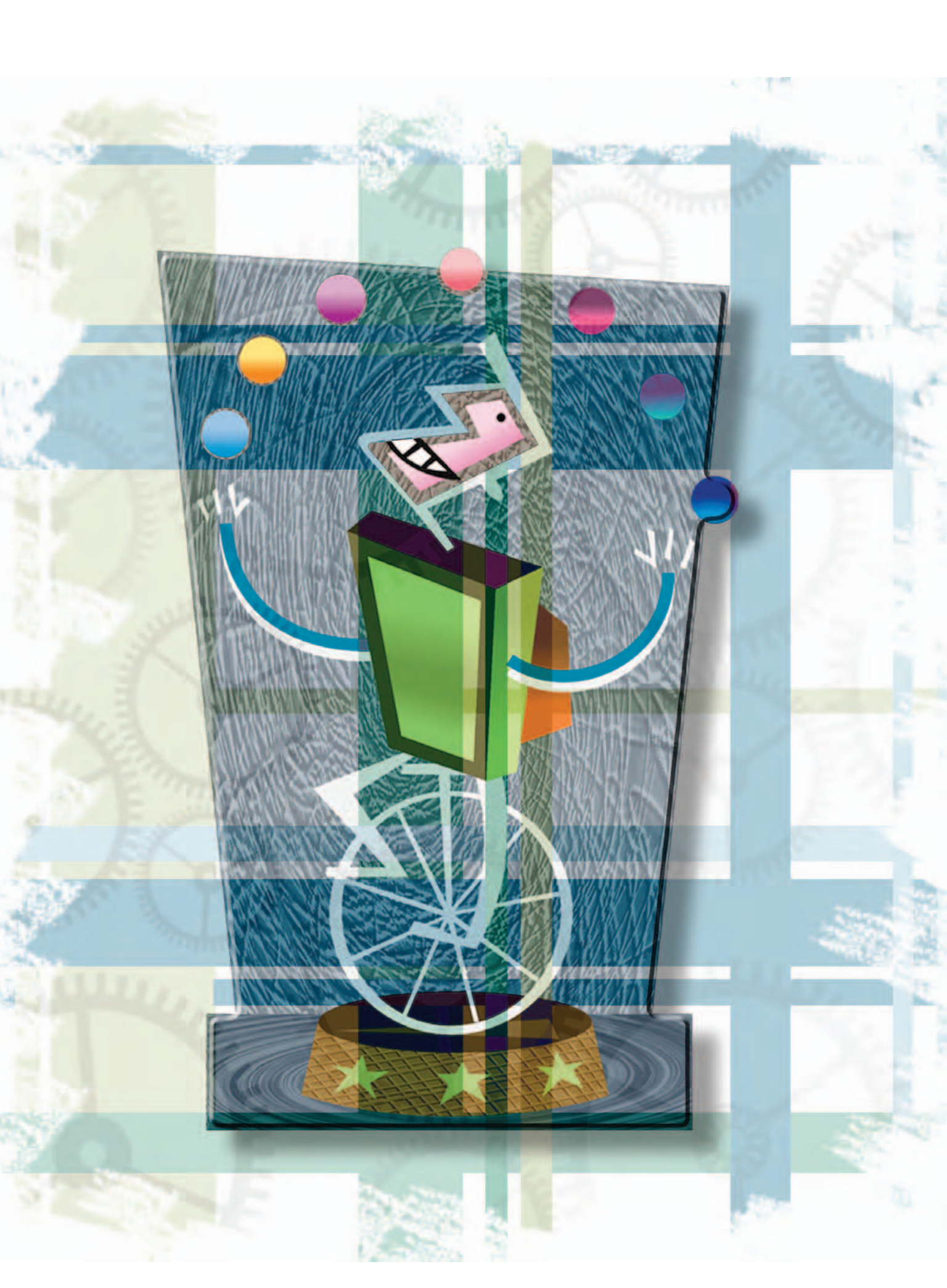
BY GAMAL ALI LABIB

Applying multitasking to an embedded system based on a PIC microcontroller may look appealing. It permits full use of the PIC resources by serving more than one task at a time rather than staying idle waiting for an external event to occur and reacting accordingly from within an application. However, multitasking can turn applications into memory-hungry monsters that barely fit in your PIC. Task reentrancy is one approach that tackles expanded memory tasks, especially in small to medium PICs.

In this, the first part one of a three part series about reentrant multitasking and how you can realize it with two working examples of a Microchip's PICmicro MCU, I begin with an introduction to multitasking. In part two and three, available on Embedded.com, I'll get more into the meat of multitaskgin on the PIC.

Embedded multitasking is not new. It goes back some 30 years. The Intel 8080, Zilog Z80, and Motorola 68000 were popular microcontrollers 30 years ago. Interestingly, they were very similar in processing capabilities to many of today's microcontrollers. Those ancient devices had multiple general-purpose and index registers that could perform arithmetic and logic operations, and supported vectored interrupts and I/O ports. They had data buses with widths up to 16 bits, and address buses up to 24-bits wide. They also had large and powerful instruction sets that gave way to efficient programming and ran at moderate speeds of around 10 MHz. What's changed over the years is that today, that same processor is packaged with RAM and EPROM. Adding analog-to-digital modules and EEPROM gives us today's "microcontrollers."

Thirty years ago, full-fledged real-time operating systems (RTOSs) existed and professional features like multitasking and



multiprocessing were applied and implemented. This brings up an interesting point: where exactly does multitasking fit with this technology? With a little research, I found that products are available to handle multitasking with an RTOS. Developers must then question whether the RTOS is really required for their application, or near-real-time would suffice.

Back then, in the absence of multitasking, serial task execution was the only way to run embedded systems. Figure 1 shows two tasks running in a system. Task #1 executes until it concludes its operation, then passes control to Task #2. Task #2 runs and eventually terminates causing system operation to end.

By definition, multitasking allows an embedded system to share its resources between several programs running concurrently. The resources can be those of the processor itself (such as internal registers, memory, and arithmetic logic units) or the peripherals attached to the embedded system (such as display, printer, sensors, and actuators).

Figure 2 illustrates a simple form of multitasking. Here, Task #1 is the main task running in the system until some sort of interrupt or an event occurs at time *t*1. This causes the system to put that task on hold, passing its control to another secondary Task #2, at time *t*2, to serve the event and willingly return system control to the main task at time *t*3. This kind of multitasking is based on cooperation between the running tasks, such that tasks are willing to give up system control at some point throughout their execution. Note that there's no mention of operating-system coordination between tasks; task delegation of system control is built in the task programs themselves. Nested subroutine calls and branching instructions are a means of implementing simple multitasking.

Back then, in the absence of multitasking, serial task execution was the only way to run embedded systems.

Multitasking in an operating-system environment is a different story. System resource sharing is realized under the operating system's control, rather than task delegation, by allocating those resources to a single task for a specific period of time before putting them temporarily on hold and switching the resources to another task. The task allocated time and frequency depend on "multitasking policy." Different policy schemes can be adopted, such as round-robin (equal task rights) and priority-based (higher priority tasks use resources more frequently than lower priority ones).

COOPERATIVE VS. PREEMPTIVE MULTITASKING

The part of the operating system responsible for task switching is called the *task scheduler.* The scheduler enforces the multitasking policy whenever it regains system control from the user tasks. User tasks can be designed to either give up system control to the scheduler voluntarily (scenario #1) or forcibly (scenario #2).

Scenario #1, called *cooperative multitasking*, is where a breakpoint is implanted in the task program to

Two tasks are running in a system.

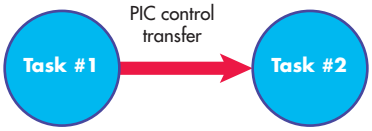


Figure 1

A simple form of multitasking is shown.

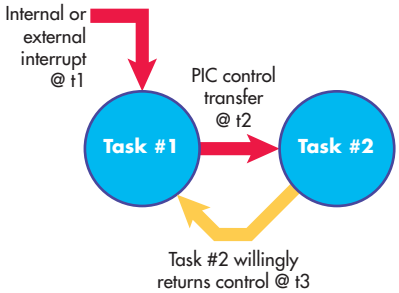


Figure 2

As shown, the process gets more complicated when three tasks are implemented. This is called cooperative multitasking.

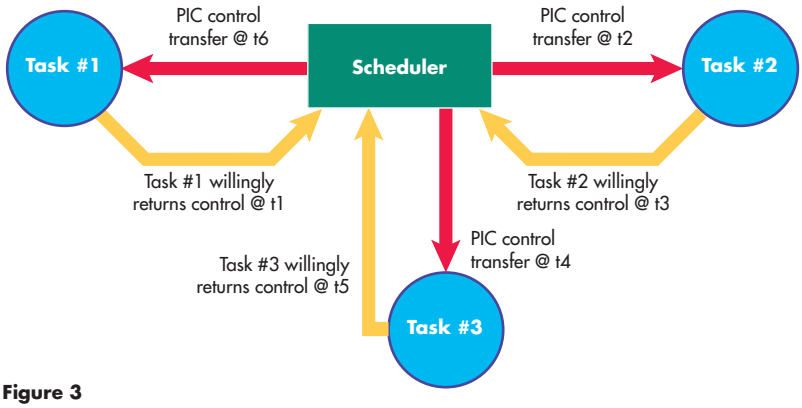


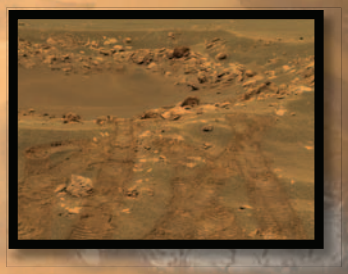
Figure 3

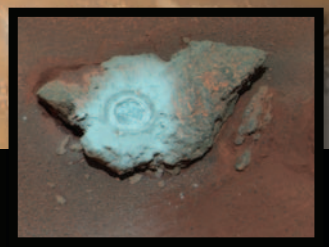
SMILE, MARS!

ThreadX® RTOS manages camera software critical to NASA mission

"We found ThreadX to be a proven solution based on its demonstrated success for the Deep Impact mission, so using it for the HiRISE instrument aboard the MRO was a logical decision. ThreadX delivered a first-rate performance for us and helped the MRO mission return extraordinary high-resolution images from Mars."

– Steve Tarr, HiRISE Software Lead, Ball Aerospace & Technologies Corp.





MRO spacecraft depicted in Mars orbit: NASA

Images courtesy NASA: http://mars.jpl.nasa.gov/mro

The Mission

When they wrote the embedded software that controls the cameras aboard the Mars Reconnaissance Orbiter (MRO), a team of Ball Aerospace and Technology Corp. engineers led by Steve Tarr knew they only had one chance to get it right. If there was a serious flaw anywhere in the software, the $720 million spacecraft might have no more value than a digital camera dropped in a bathtub.

THREADX

Tarr and his team wrote 20,000 lines of code and used Express Logic's ThreadX RTOS. The software has worked flawlessly, resulting in history-making photographs such as the one to the left that shows the Opportunity rover traversing the surface of Mars.

The Technology

With its intuitive API, rock-solid reliability, small memory footprint, and high-performance, ThreadX delivered the goods for NASA's MRO. ThreadX is in over 450 million electronic devices from NASA's MRO to HP's printers and digital cameras. Which RTOS will you choose for YOUR next project?

Small Memory Footprint • Fast Context Switch • Fast Interrupt Response
Preemption-Threshold™ Technology • Picokernel™ Design • Event Chaining™
Broad Tools Support • Supports All Leading 32/64-bit Processors • Easy to Use
Full Source Code • Royalty-Free

Order today on Amazon
Real-Time Embedded Multithreading
Using ThreadX and ARM
by Edward L. Lamie

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

expresslogic

transfer system control to the scheduler program. The scheduler can later resume task execution at that breakpoint. Figure 3 illustrates this scenario with three tasks. Task #1 returns control to the scheduler at time *t*1. The scheduler decides that Task #2 gets the control, which is passed on to it at time *t*2. At time *t*3, Task #2 returns control to the scheduler and Task #3 gets control at *t*4. Task #3returns control to the scheduler at *t*5; the scheduler then decides that Task #1 resumes its execution. Task #1 regains control at *t*6. This scenario repeats until all tasks conclude their operation or the system shuts down.

Scenario #2, called *preemptive multitasking*, is where task execution is interrupted at any point in time by an internal or external system event. System control is then handed over to the scheduler to decide which task is to resume execution. Figure 4 illustrates this scenario with two tasks. Here, Task #1 runs until an interrupt occurs at time *t*1, suspending the task. Control is passed on to the scheduler program at *t*2, at which time the program decides to pass the control to Task #2 based on some policy and activates that task at *t*3. Task #2 runs until *t*4, when another interrupt occurs causing it to suspend and the control is passed on the scheduler again at *t*5. The scheduler decides to resume Task #1 execution at *t*6, which takes back control until another interrupt occurs. Frequent interrupts to running tasks enable the scheduler to manage system control switching between tasks.

MULTITASKING REENTRANT PROGRAMS

The tasks employed for a multitasking embedded system can be replicate control applications or a combination of control, data crunching, and user interfacing. But we know that not all embedded systems have an abundance of memory that fits multiple task programs. In addition, many PICs have limited interrupts, work registers, and stack capabilities that keep them form competing against full-fledged processors. This may explain why it's hard to find operating systems that support multitasking.

One way around the PIC's limitations allows better manipulation of its resources. In computer jargon, it's called reentrant programming.

One way around the PIC's limitations allows better manipulation of its resources. In computer jargon, it's called *reentrant programming*. Reentrant programs realize a simple form of multitasking while minimizing memory overhead. When you consider running a number [x] of tasks of similar functionality, each of which requiring [y] memory bytes to run, you may think at first glance that you would need [x * y] bytes of memory. However, with reentrancy, you would actually require only a fraction [?] of memory over and above that of a single task, giving a total of [(1 + ?) * y] bytes for the [x] tasks.

Figure 5 illustrates the reentrant program concept as a special case of multitasking, which was demonstrated in Figure 4. Here, all tasks are similar

Using preemptive multitasking, task execution is interrupted and system control is handed over to the scheduler to decide which task should resume execution.

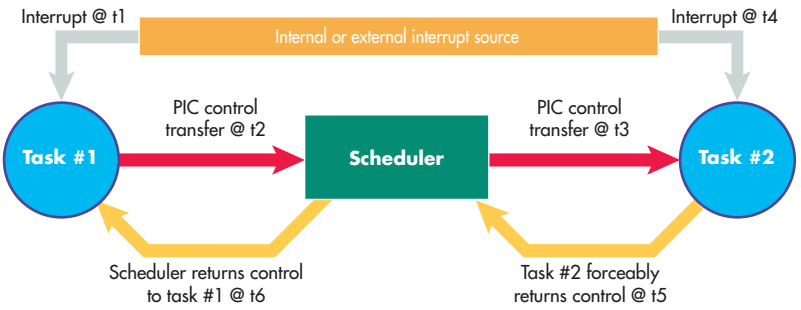


Figure 4

The reentrant program is a way to handle multitasking.

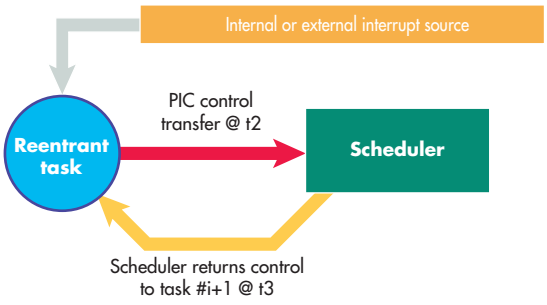
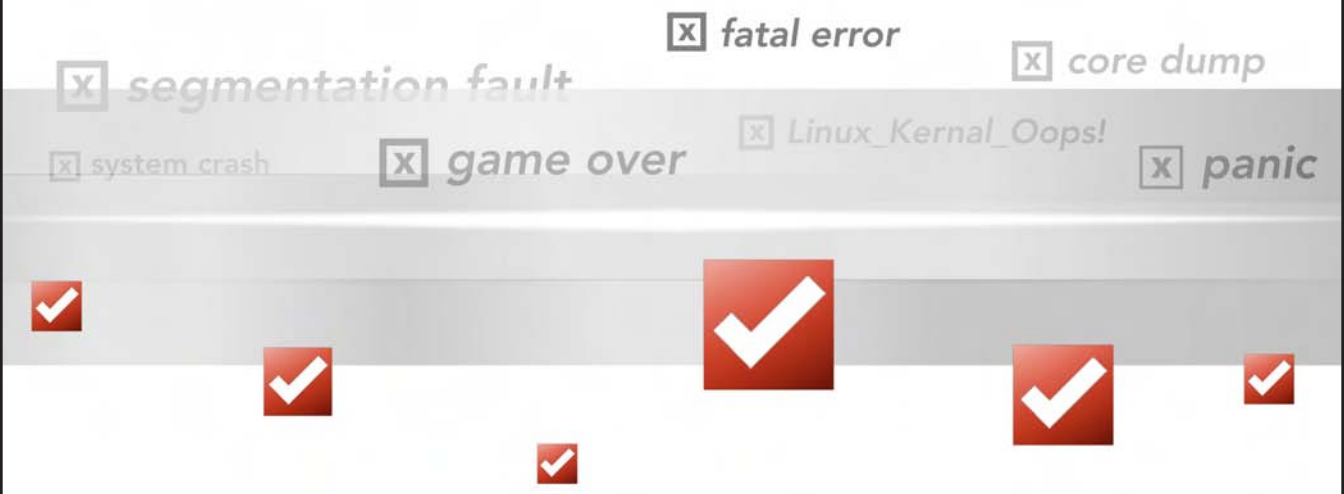


Figure 5

Choosing Linux as your next device operating system?

CHAOS



CLARITY

Wind River On-Chip Debugging offers the most advanced development and debug solution for Linux developers. Debug all the components of a Linux system simultaneously: boot loader, kernel, kernel modules, user mode applications, and shared libraries. Connect seamlessly through JTAG to a running system to gain immediate visibility into the entire Linux system. No instrumentation of the kernel needed. No need to use agent for debug.

Move from chaos to clarity. Learn how Wind River helps speed development of Linux devices. Find out more and register for Linux debugging classes in your area at www.windriver.com/clarity

WIND RIVER

in functionality and are represented by a single reentrant program. The scheduler switches the execution of this program among the distinct tasks upon interrupt events.

A reentrant program is a generic data-driven task. If you manage to separate the task data from the code and make the code register-dependent, you can end-up with a generic task. Having done this, another question arises: how can user tasks invoke their reentrant program simultaneously? The answer is twofold:

- First, you must initialize the registers of the generic task with each task's data.
- Second, you need some mechanism, such as interrupts, to switch the generic task between user tasks.

A user task gaining control of the generic task loads its processor state (its registers' settings) and resumes its previously interrupted operation. But that only happens after it the user task saves its predecessor state. The task in control of the processor may set a pointer to its descendant task to take over control when the interrupt occurs.

. . . the PIC lacks support for automatically saving and restoring its state on interrupts. This makes interrupt processing extremely delicate.

SYSTEM LIMITATIONS

My first encounter with microcontrollers was the Microchip's PIC18F452 MCU. It was a challenge to implement my version of multitasking on such a device. I was faced with limitations that required a workaround that costs memory and execution time. I didn't encounter such limitations while working with those ancient microprocessor devices.

On one hand, most of the PIC data transfer and arithmetic instructions involve the W register. To facilitate indirect addressing on a PIC, the address must first be loaded into the FSR register. This makes a stack implementation inefficient, comprising many instructions for one indirect memory access. The programmer has to work around this by passing parameters in fixed memory locations.

The PIC's small hardware return stack allows only minor nesting of calls, so the program structure is almost flat, relying on macros. Also, the PIC lacks support for automatically saving and restoring its state on interrupts. This makes interrupt processing extremely delicate. It's up to the programmer to save the processor state on interrupt, but this entails a strict instruction sequence.

Figure 6 shows how the memory in a PIC-based embedded system is allocated for a multitasking environment. Two cases are shown for the program memory: the memory map on the left shows a multiple of similar user tasks with replicated code. The map on the right uses a reentrant code. Common program modules in both cases are the interrupt handler, the task initialization, and the task scheduler. Those modules are almost identical in both cases. The data memory holding the tasks' work areas is also similar but with minor differences. The software stack in the data memory solves the drawbacks of the PIC's hardware stack. ■

The figure shows how the memory in a PIC-based embedded system is allocated for a multitasking environment.

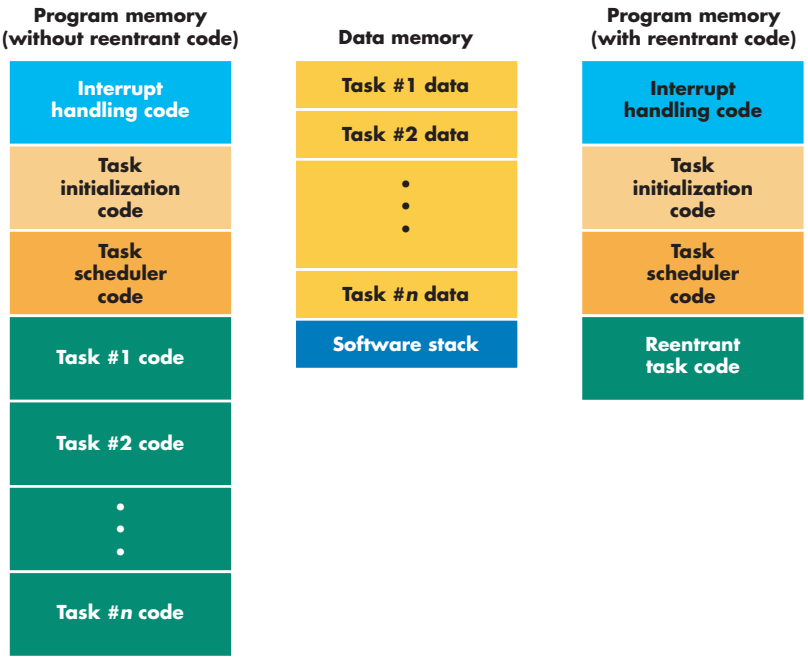


Figure 6

Gamal Ali Labib is an IT consultant in Cairo, Egypt. He specializes in IT security and IT turn-key projects management. He is also interested in parallel processing and VLSI. Dr. Labib has a B.Sc. and M.Sc. in computer engineering and electronics from Ain Shams University, Egypt, and a PhD in computer science from University of London, U.K. You can reach Dr. Labib at drgamallabib@yahoo.co.uk..

Boldly go where no system has gone before....

EPIA PX-Series Pico-ITX Mainboard

- Revolutionary Form Factor for Embedded Applications
- VIA C7 1.5GHz Processor
- Pin Header I/O for System Design Flexibility

USB 2.0

15 Independent 16-Bit Counter / Timers

USB 2.0

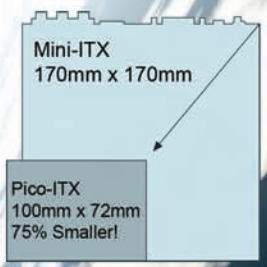
16 Isolated Digital Inputs 16 Relay Outputs

USB 2.0

16-Bit Analog Input 16-Channels 500kHz

USB 2.0

8-Channel 125kHz Waveform D/A Output



VIA Technologies' tiny PICO SBC combines with ACCES I/O Products' PC/104 size embedded USB boards for OEM data acquisition and control.

With 4 USB ports, dozens of USB/104 I/O modules to choose from, and extended temperature options - Explore the Possibilities!

Saving Space, The Final Frontier

VIA embedded in your success

To find more information on EPIA mainboards, please visit
www.VIAembedded.com
or call 510 683 3300. Come visit us at
940 Mission Court Fremont CA 94539

ACCES I/O PRODUCTS, INC.
The source for all your I/O needs

To learn more about our Embedded USB/104 I/O boards visit
www.accesio.com
or call 800 326 1649. Come visit us at
10623 Roselle Street San Diego CA 92121

VIA C7 Processor
VIA VX700 IGP Chipset
ROHS COMPLIANT

These key tools can be fine-tuned to fit your needs and your style of work.

Do-it-yourself Linux development tools

BY ALEXANDER SIROTKIN

Development tools are important. They save development and debug time. But most importantly, they make developers more happy and productive by automating many routine, boring, and time-consuming tasks. It's painful to see programmers spend a significant percentage of their valuable time on such routine tasks as downloading their code to the embedded target. This situation is not uncommon even with traditional embedded systems, but it's far worse with embedded Linux, where the lack of good development tools is evident.

The perception that there are no good embedded Linux tools is not entirely true. First, a few commercial tools are available that are worth considering. However, contrary to the traditional embedded systems development world, commercial tools aren't the only option. A lot can be done with relatively little effort using freely available open-source tools. In fact, I would argue that the do-it-yourself development environment can be far superior to any commercial offering, as it will allow you the level of customization, flexibility, and agility that none of the off-the-shelf products can match.

COMMERCIAL TOOLS

Embedded Linux integrated development environment (IDE) software suites are usually available from the same companies that sell embedded Linux. Wind River, MontaVista, TimeSys, LynuxWorks, and a dozen other vendors come to mind. Although these companies will try to sell you both the operating system and the accompanying IDE, this IDE may or may not be tied to that particular distribution. Most of these tools are Eclipse-based and offer similar

embedded

functionality, which at closer look turns out to be no more than a clumsy editor, compiler wrapper, and debugger. The only exception I'm aware of is Wind River Workbench, which is actually a commercial grade product—not surprising considering Wind River's experience with its Tornado IDE for VxWorks.

The major problem with off-the-shelf IDE suites isn't the software itself, but rather the nature of embedded systems development. To unleash the full power of an IDE, you must run an agent on the target embedded platform, as illustrated in Figure 1. This agent, however, may not be available if you're working on a customer system, or you may not have enough time to integrate it if you're doing a relatively short-term project. However, this agent typically fails to run because it relies on some kernel functionality that may not be available, as all embedded platforms are different and embedded systems programmers love to tweak system internals, often breaking some functionality the IDE agent relies on.

DO IT YOURSELF

The do-it-yourself approach has many advantages, such as the resulting tool is free, not tied to any particular Linux distribution, customized for your needs, and most importantly, it's modular. This lets you quickly port to a new platform only the functionality that you really need.

Now, let's take a look at the most common development tasks and see how they can be automated and simplified using open-source tools. We'll also try to bind it all together to form something similar to IDE, although it's not really necessary—every tool described here can be used by itself. Note that all network-related examples assume that all networking components reside on the same LAN with a 192.168.0.0/24 subnet; 192.168.0.9 is assumed to be host address and 192.168.0.10 the target.

HOST PLATFORM

Although everything I will describe can be also done on a Windows host, I recommend using Linux. It's more convenient, and more tools and utilities are available. And if you rely on a few Windows applications such as Word and Outlook, you can still run them on Linux in emulation using VirtualBox, Wine, or other commercial package. If you're new to Linux, using a Linux host will also force you to learn the new platform faster.

WRITING CODE

Many developers have definite preferences with regard to programming editors, and there are many open-source and commercial Linux packages to choose from. Emacs is my favorite, as it has all the required features (and much more) and most probably will be installed on every Linux host.

Emacs is a powerful editor, but it requires a bit of configuration to unleash it's power. I recommend enabling at least the following options: global font-lock mode for syntax highlighting, imenu-add-menubar-index to show a list of C functions defined in the current file as a drop-down menu, and cscope (which is a separate package with Emacs integration support) tags for fast search and code browsing. You can either change individual options through Emacs' Options menu, or just cut and paste the example in Listing 1 into your .emacs file.

I recommend running the compilation process inside Emacs. Having the following keyboard shortcut will make it more convenient:

```
(global-set-key 'f5 'compile)
```

Emacs has many other features that a programmer would find handy, such as version control integration, autocompletion, matching parentheses, auto-indentation, C macro expansion, gdb debugger integration, and code folding (for further reading, I suggest "Emacs for programmers," *www.linuxjournal.com/article/2821* and "Using Cscope and SilentBob to analyze source code," *www.linux.com/article.pl?sid=07/03/05/1715201*). And don't forget to print the Emacs reference card (refcard.ps) that came with your Emacs installation.

RUNNING YOUR APP

Downloading and running your code on the embedded target is probably one of the most frequent operations you'll do, so it's important to make this

To unleash the full power of an IDE, you must run an agent on the target embedded platform.

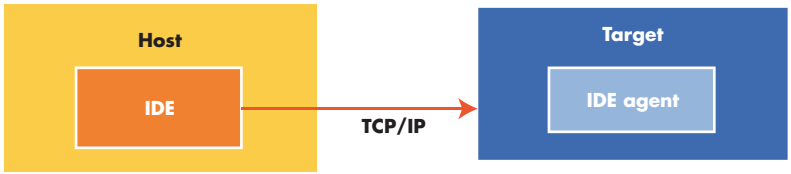


Figure 1

Listing 1 Cut and paste this code into your .emacs file to enable useful features.

```
(global-font-lock-mode t )
(setq cscope-do-not-update-database t)
(setq imenu-sort-function (quote imenu--sort-by-name))
(require 'xcscope)
(add-hook 'c++-mode-hook 'imenu-add-menubar-index)
(add-hook 'c-mode-hook 'imenu-add-menubar-index)
```

process as fast and easy as possible. This usually done using TFTP or JTAG, but Network File System (NFS) can make this process easy and transparent. NFS can be used by running the NFS server on your host and having makefile rule to copy the compilation results to the NFS root directory and mounting this directory via NFS from the embedded target. This way, the compiled application will appear immediately at the target without any intervention by the programmer, as illustrated in Figure 2.

When implementing the host configuration, make sure that the NFS server is installed, and add the following line to /etc/exports (after modifying IP and directory):

```
/home/user/nfsroot 192.168.0.*
    (rw, no_root_squash, sync)
```

then restart the NFS server. This gives NFS clients from 192.168.0.0/24 subnet full read/write access to your nfsroot directory.

In the target configuration, make sure that the kernel is compiled with NFS support. In other words, CONFIG_NFS_FS, CONFIG_NFS_V3 and CONFIG_NFS_V4 options are enabled and the mount command supports NFS (if you are using busybox, check that the CONFIG_FEATURE_MOUNT_NFS option is enabled). You'll have to issue the following command on the target machine (probably as part of initialization script):

```
mount 192.168.0.9:/home/user/
     nfsroot /mnt/nfs.
```

If you can't use NFS, whether because you can't modify the kernel or because your short term project doesn't justify the effort to configure it, you still don't have to copy your application manually from host to target. This task can be automated using expect or minicom scripts, as shown below.

AUTOMATION

Many tasks can be automated using shell scripts. Note that you can run scripts on the host as well as on an embedded target (most embedded Linux distributions include shell with scripting functionality, although it's more limited). An introduction to bash scripting can be found at *www.linuxjournal.com/article/1299*, but remember that you won't be able to use many of the advanced scripting options on your embedded target, as it'll likely have one of the less powerful (and less memory hungry) shells, such as BusyBox ash.

Using interactive expect(1) scripts, you can automate such tasks as image download and flash programming.

Assuming you're capable of writing a simple bash script, we'll dive into a bit more complex but useful topic, interactive scripts. Using interactive expect(1) scripts, you can automate such tasks as image download and flash programming. But you can't use simple bash scripts to automate interactive tasks because of branching and timing issues.

Basic expect(1) scripts consist of a "spawn" command that executes the utility that requires interactive automation, such as telnet or minicom and a series of "expect" and "send" commands, as illustrated by the example in Listing 2.

The `expect` command waits until one of the patterns matches the output of a spawned process. `Send` sends string to the current process. The script in Listing 2 downloads and copies a new bootloader image to flash. It's written for Das U-Boot, but can be easily modified for any other bootloader or other environment. Most expect(1) distributions include a handy autoex-

pect script that automatically remembers the commands you type and creates an expect script for you. However, I encourage you to use automatically generated scripts only as a template for writing your own, which will be more readable and easier to maintain.

DEBUGGING

Debugging embedded Linux is tricky, because the technique can be different depending on whether you are debugging applications, drivers, or kernel code. The only common element is the gdb client front end, so we'll start there. A gdb client will usually run on the host platform, although technically you can run it on the target, too. It can connect to the gdb server running on the target (more on that later) through a serial port or tcp/udp protocol. Note that you can't use the x86 gdb client that comes with your Linux distribution. You'll need the one from the cross-compiler toolchain for your embedded CPU.

Simple gdb debugging session is illustrated as:

```
gdb>
file vmlinux
target remote 192.168.0.10:2828
Ctrl-C
bt
```

This example loads `vmlinux` image symbols, connects to a remote target, interrupts the running code, and prints backtrace (for more information about gdb commands, see the gdb user manual at *sourceware.org/gdb/current/onlinedocs/gdb_toc.html*). Using a command-line interface is handy for quick tasks, but for serious debugging, most users prefer a graphical interface. I recommend two graphical gdb wrappers, DDD and Insight.

Using NFS to download image to embedded target.

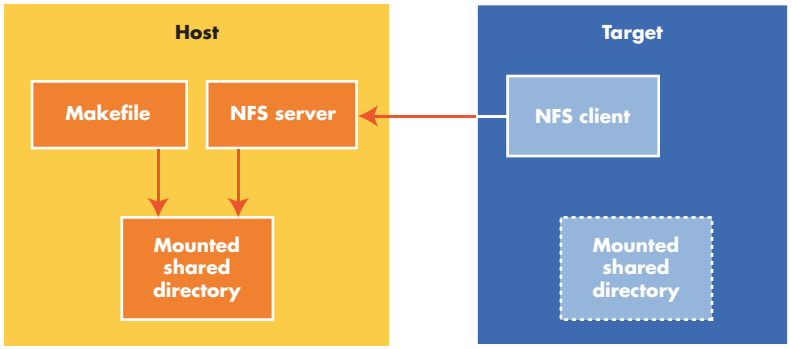


Figure 2

Insight has a slicker user interface, but because it incorporates gdb, you'll need a different binary for every embedded CPU architecture you work with. DDD is a bit more clumsy as it's a GUI wrapper, but it can work with an external gdb executable (using the `–debugger` parameter) allowing you to use a gdb binary from your cross compiler toolchain. The gdb server will vary depending on what you're debugging. For applications, you'll need to run a gdbserver executable on the target in the following way:

```
gdbserver 192.168.0.10:2828
    your_application
```

`gdbserver` will run the binary `your_application` that you're going to debug and wait for gdb client to connect on the specified port. It can also attach to a running process if executed with `–attach` command-line argument.

Kernel debugging is trickier for various reasons, not the least of which is the fact the kgdb (kernel gdbserver equivalent) may not be integrated into your kernel, so you may need to download a kgdb patch from *kgdb.linsys soft.com*, apply the patch, and recompile the kernel. Doing this will allow you debug the kernel through a serial port or over Ethernet. When debugging kernel loadable modules using kgdb version 1.8 and earlier, you'll have to load a module object file into gdb manually using the loadmodule.sh

Listing 2 Basic expect(1) scripts consist of a "spawn" command that executes the utility that requires interactive automation, such as telnet or minicom, and a series of "expect" and "send" commands.

```
#!/usr/bin/expect
spawn minicom
send "\r"
expect ">" { send -s "tftp 0x1000000 u-boot.bin\r" }
expect "done" {send -s "protect off 0x20000000 0x2007ffff\r" }
expect "Un-Protected" {send -s "erase 0x20000000 0x2007ffff\r"}
expect "Erased" { send -s "cp.b 0x1000000 0x20000000 0x20000\r" }
expect "done"
```

script or add-symbol-file gdb command so that gdb will be aware of your module's the symbol table.

If you're creating a board support package or doing some low-level kernel programming, you'll probably need a JTAG probe. When choosing a probe, ensure that it supports Linux, although most probes nowadays do. A Linux-friendly JTAG probe should support the Linux target and host; support a remote gdb protocol for debugging; support the debugging of kernel code, applications, and dynamically loadable modules; and support a Linux MMU.

The latter one is tricky because during debugging, you may want to access a virtual memory page that's not currently mapped. The probe will have to either know how to extract this mapping information from Linux internal data structures or make Linux remap this virtual address. Linux host and remote gdb support is not essential, but it is convenient, as it lets you debug with JTAG using the same gdb client frontend.

TRACING

Debugging lets you to catch simple bugs, but unfortunately the hard ones are usually affected by timing and probably won't show up in the debugger. If this is the case, you consider using the Linux Trace Toolkit (LLT), which can be downloaded from *www.opersys.com/LTT*. It allows tracing various event types for multiple processes and the kernel itself, and it can present this information graphically to help debug complex multiprocess systems. To use LTT, you'll have to patch the kernel and install LTT daemon and utilities. Note that LTT daemon requires read-write filesystem access to store the trace file. I don't recommend using the jffs2 filesystem to store such information in flash as this could severely effect system performance and timing. A RAM disk is the best option, provided you have enough memory. If this isn't the case I suggest using NFS.

The most simple LTT session looks like:

```
trace 60 trace_file
traceview trace_file
```

Both commands are a helper scripts. The first enables tracing for 60 seconds and uses `trace_file` as a base name for the trace results file. The second executes the graphical trace visualization tool (for more information, read the LTT reference manual at *www.opersys.com/LTT/dox/ltt-online-help/index.html*).

. . . embedded developers . . . can filter profiling results by modules, which can be a driver, an application, or the kernel itself.

PROFILING

Eventually all embedded systems projects reach the optimization phase where a good profiler can come in handy. OProfile is a systemwide profiler for Linux that's capable of profiling user space applications, kernel code, and loadable modules. The OProfile software package consists of kernel code, userspace daemon, and utilities. Fortunately, no kernel patching is required as OProfile is part of the stock Linux 2.6 kernel.

A typical OProfile session looks like:

```
opcontrol –vmlinux
    =/path/to/vmlinux
opcontrol –start
#execute your code
opreport -l /path/to/mybinary
opcontrol –reset
```

The first two commands start the profiler, opreport prints the results, and the last command resets the data. OProfile is a powerful profiler with many features, all of which are outside the scope of this article, but I encourage you to read more at *oprofile.sourceforge.net/doc/index.html*.

An important tip for embedded developers is that they can filter profiling results by modules, which can be a driver, an application, or the kernel itself. Note that OProfile is a statistical profiler, which means it samples program-counter values on interrupt (usually timer interrupt), so you must run the code for some minimal time period to get meaningful results.

Listing 3 Emacs Lisp code that should be added to your .emacs file to create IDE-type keyboard shortcuts.

```
(define-key global-map [menu-bar embedded]
 (cons "Embedded" (make-sparse-keymap "Embedded")))
(define-key global-map
 [menu-bar embedded program-flash]
 '("Program flash" . program-flash))
(define-key global-map "\C-cpf" 'program-flash)
(defun program-flash ()
 "Program flash."
 (interactive)
 (setq embedded-buffer
 (get-buffer-create "Embedded Buffer"))
 (display-buffer embedded-buffer)
 (set-buffer embedded-buffer)
 (setq buffer-read-only nil)
 (call-process "program-flash.sh" nil embedded-buffer t)
```

If your hardware is like most, and has performance counters that can generate an interrupt on such events as cache miss, you can find not only the most CPU-intensive pieces of code, but also the most cache inefficient. Unless you use an exotic CPU, OProfile will already have support for performance counters of that CPU.

BINDING IT ALL TOGETHER

Although it's not necessary and can even make things more complex, if you're more comfortable with a monolithic IDE-like software package that can do all your daily tasks using keyboard shortcuts and menus, the simplest way to do this is to create a new Emacs menu called "Embedded" and add all your scripts to that menu. The example in Listing 3 (Emacs Lisp code that should be added to your .emacs file) illustrates this.

Full explanation of this code requires some Lisp knowledge (and beyond the scope of this article), but a brief description of each command will be given, so you can customize the code and add new functionality without a full understanding of Lisp and Emacs internals. The first set of commands creates the "Embedded" drop-down menu. The second one adds a new entry "Program flash" to the menu that executes the "program-flash" Lisp function. The third set binds that command to "Ctrl-c" followed by a "pf" key combination. The final one is a Lisp function that executes a "program-flash.sh" shell script redirecting its output to a newly created Emacs buffer.

Creating an embedded Linux development environment by yourself may look like a serious task. But keep in mind that this is a good investment. This task can be easily outsourced to an embedded Linux consultant, in which case it'll probably cost less than an off-the-shelf embedded Linux IDE.

Note that in the Linux world, there many different ways to accomplish the same task. This article shows just one of those alternatives. ■

Alexander Sirotkin works for Metalink Broadband as a software architect. For more than 10 years, he's been dealing with software, operating systems, and networking, and holds M.Sc. and B.Sc. degrees in applied statistics, computer science and physics from Tel-Aviv University. Sirotkin can be reached at sasha.sirotkin@gmail.com.

RESOURCES

Wine—*www.winehq.org/*
VirtualBox—*www.virtualbox.org/*
Eclipse—*www.eclipse.org/*
Emacs—*www.gnu.org/software/emacs/*
Cscope—*cscope.sourceforge.net/*
Busybox—*busybox.net/*
GDB—*www.gnu.org/software/gdb/*
DDD—*www.gnu.org/software/ddd/*
Insight—*sourceware.org/insight/*
OProfile—*oprofile.sourceforge.net/*

BY Danny R. Graves

Search for embedded Linux patents

As an electrical engineer with an automotive background, when I think of Linux, I think of servers, PCs, supercomputers, and so forth. Embedded applications don't really come to mind when I consider Linux. However, Linux is used as an operating system for many phones, games, and other devices with embedded software.

Computer programs, often protected by copyright or trade secrets, can't be directly patented unless they're used for something tangible, such as signal processing or hardware control. For example, an operating system could be patented as a business method or a method to control computer hardware. Even though Linux is open source (free), certain companies could have patents that could be infringed by people using Linux in embedded applications.

Linux is generally considered free software, but is its use in embedded devices protected by U.S. patents? A patent consists of several parts, including the abstract, specification, and claims. The *abstract* is a concise summary of the specification while the *specification* is a complete description of the invention. The *claims* are where the majority of the legalese is found and are generally difficult for a nonlawyer to understand. Reviewing patent specifications and claims can give insight into the popularity and application of certain technologies throughout the years.

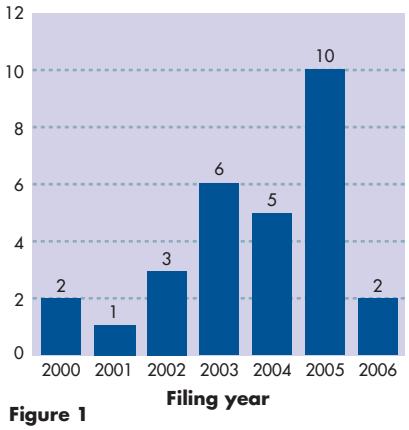


Figure 1
Number of U.S. patent documents related to embedded Linux per filing year.

In many cases, "free" isn't really free.

The claims are the most important part of a patent from a legal point of view because they define exactly what part or aspect of the invention is patented and, therefore, legally protected. From a technical point of view, the specification is often the most useful as it's a complete technical description of the invention and usually has less of the legalese that's found in the claims. If a feature is found in the claims, it'll definitely be in the specification. However, a feature may be discussed in the specification but not in the claims.

PATENT OFFICE DATABASES

The United States Patent and Trademark Office (USPTO) makes a lot of patent information available online at *www.uspto.gov/patft/index.html*. You can actually look at patents dating back to 1790 (Abe Lincoln's patent #6,469 for "A Device for Buoying Vessels Over Shoals," on May 22, 1849, is in the online database!). Published patent applications can also be searched and viewed using the site. Patent applications are usually published on the site 18 months after they're filed. Batches of applications are published every Tuesday.

Patents are assigned to certain technical classes based on the nature of the invention. Each class has multiple subclasses that can be used to further specify the type of invention. You can see the patent classes at *www.uspto.gov/go/classification/selectnumwithtitle.htm*. As an example, patents related to operating systems might be contained in class 713, Electrical computers and digital processing systems: support; or class 719, Electrical computers and digital processing systems: interprogram communication or interprocess communication.

You can search for patents by looking in the appropriate classes and subclasses if you can determine them. However, embedded Linux applications could be located in many different classes because they can be classified by the end system's application.

Another way the databases can be searched is by keywords in the abstract, specification, or claims. The databases can also be searched by inventor, assignee, filing date, or several other parameters. You can combine the different search parameters logically using AND, OR, and the appropriate parentheses.

FINDING EMBEDDED LINUX

I searched the issued patent database, on 10/26/07, for titles, specifications, claims, or abstracts containing "linux" and "em-

Danny Graves is a freelance patent analyst, patent agent, engineer, college instructor, author, and inventor with two U.S patents. He was a finalist for the 2001 Charles C. Gates Award for Excellence. He has researched numerous inventions across a wide range of technologies. Graves can be reached at dannygraves@bellsouth.net.

Partial list of U.S. patents and patent publications related to embedded Linux as of 10-26-07.*

Patent or Publication Number	Title	Filing Date	Assignee
US20070101112	Embedded device detecting system and related method	10/27/05	INVENTEC CORPORATION
US20060190939	Updatable mobile handset based on linux with compression and decompression techniques	12/21/05	BITFONE CORPORATION
US20060190933	METHOD AND APPARATUS FOR QUICKLY DEVELOPING AN EMBEDDED OPERATING SYSTEM THROUGH UTILIZING AN AUTOMATED BUILDING FRAMEWORK	07/18/05	LITE-ON TECHNOLOGY CORP.
US20060167910	[ELECTRONIC DEVICE WITH AN EMBEDDED LINUX APPLICATION SYSTEM]	01/25/05	Not Available
US20060010314	Methods and systems for running multiple operating systems in a single mobile device	07/07/04	Not Available
US20050050242	Method of safely shutting down an embedded system based computer under Linux	08/27/03	INVENTEC CORPORATION
US20040244008	Method of implementing Linux-based embedded system for mobile communication	03/10/04	PANTECH CO., LTD
US20040133720	Embeddable single board computer	12/10/03	AMC TECHNOLOGIES CORPORATION
US20040021681	Dual-touch-screen mobile computer	09/24/02	Not Available
6886060	Computer system for integrating car electronic devices	03/21/03	INDUSTRIAL TECHNOLOGY RESEARCH INSTITUTE

Table 1

**Full list is available with online version of this column on Embedded.com.*

bedded" using the search logic ttl/(LINUX and embed$) OR abst/(LINUX and embed$) OR ACLM/(LINUX and embed$) OR spec/(LINUX and embed$) where $ represents a wildcard. This resulted in 1,445 hits. However, this included many patents discussing such things as embedded URLs and embedded images, which aren't really of interest for our topic.

By eliminating the specification from the search and limiting it to the title, abstract, or claims, I reduced the chaff, leaving only patents strongly related to Linux and embedded systems. Again, I searched the issued patent database for titles, claims, or abstracts containing "linux" and "embedded" using the search logic ttl/(LINUX and embed$) OR abst/(LINUX and embed$) OR ACLM/(LINUX and embed$). This resulted in just five hits.

I searched the pending patent publication database for titles, specifications, claims, or abstracts containing "linux" and "embedded" using the search logic ttl/(LINUX and embed$) OR abst/(LINUX and embed$) OR ACLM/(LINUX and embed$) OR spec/(LINUX and embed$). This resulted in 6,289 hits. However, like the issued patents, most of these weren't of interest. So, again I eliminated the specification from the search parameters to reduce the chaff.

I searched the pending patent publication database for titles, claims, or abstracts containing "linux" and "embedded" using the search logic ttl/(LINUX and embed$) OR abst/(LINUX and embed$) OR ACLM/(LINUX and embed$). This resulted in 56 hits.

Because patent application publications are left in the database even after the patent issues, you'll find some repetition of inventions between the issued patent and patent application databases (although the application and the patent often differ substantially after the examiner gets through with them). After eliminating the replication and remaining nonrelevant patents, only around 29 patents/publications were left. Table 1 shows some examples of relevant U.S. patents and publications related to embedded Linux as of 10/26/07.

The patent documents located reveal that a variety of companies/investors are involved with embedded Linux. There isn't one company that really dominates the list and several of them are foreign companies. So, if you work with embedded Linux, searching patents can provide you with a nice target list of companies.

Most of these patent applications were filed in the 2002 to 2005 timeframe, although two were filed in 2000. Figure 1 shows a graph of the number of documents per filing year. Applications from 2006 aren't fully represented in the data due to the 18 month delay between filing and publication of the application. So the 2006 applications related to embedded Linux are likely much higher than represented by the data.

In summary, most of the patent documents related to embedded Linux located in this search were filed between 2002 and 2005. No one company dominates the list of assignees but, rather, several companies from across the world. The number of U.S. patent applications filed related to the subject has seen as generally upward trend over the last few years, indicating increased popularity. Finally, even though Linux is a free-software operating system, it would be wise to search the U.S. patent database before commercially using Linux in an embedded application. Of course, refer to a licensed patent attorney if there is any doubt. ■

Perhaps we attract more of the embedded design community because we're part of the community, ourselves.

The people behind Embedded.com, the Embedded Systems Conferences, and *Embedded Systems Design* magazine are not dilettantes or opportunists. They are engineers and technical experts who know your world because they've lived it, and understand the information you want and need to excel in your career. Perhaps that's a big reason why professionals like you have made our website, peer-to-peer conferences, and magazine the most popular in the field of embedded design. After all, we and you are both part of the same community.

Richard Nass
Editorial Director, Embedded Systems Conferences and Embedded.com
Editor-In-Chief, *Embedded Systems Design*
Drawing on two decades of experience in the electronic OEM industry, with more than nine years in the wireless portion of the market.

Jack Ganssle
Technical Editor
Noted industry speaker, author, instructor, and member of the Embedded System Conference's Advisory Board and NASA's Super Problem Resolution Team.

Colin Holland
European Correspondent
Print and online editor of *EE Times Europe* and *Embedded System Design Europe* with nearly 20 years experience as a writer and editor at CMP Electronics in the UK.

Dan Saks
Contributing Editor
One of the world's leading experts on the C and C++ programming languages and their use in developing embedded systems.

Jack Crenshaw
Contributing Editor
Senior Principal Design Engineer for Alliant TechSystems, Inc., and a sought-after expert in compiler theory, guidance and control theory.

Join thousands of other engineers who believe that learning never stops.

Embedded Systems Design Embedded.com Embedded Systems CONFERENCES

Tap the power of 3

CMP
United Business Media

ad index

Advertiser	URL	Page
ACCES I/O PRODUCTS INC	www.accesio.com	41
ALTERA CORPORATION	www.altera.com	CV4
ATMEL CORP	www.atmel.com/AVRman	29
CMX SYSTEMS INC	www.cmx.com	48
EMAC INC	www.emacinc.com/sbc-microcontrollers/ipac_9302.htm	55
EXPRESS LOGIC	www.rtos.com	37
GREEN HILLS SOFTWARE INC	www.ghs.com	1
HI-TECH SOFTWARE	www.microchip.htsoft.com	2
INTEL	http://developer.intel.com/design/info/906.htm	13
KEIL SOFTWARE	www.keil.com/demo	31
LOGIC SUPPLY	www.logicsupply.com	20
MCOBJECTS LLC	www.mcobject.com	47
MENTOR GRAPHICS	www.mentor.com/embedded	CV3
MICRO DIGITAL	www.smxrtos.com/processors	45
MICRO DIGITAL	www.smxrtos.com/fs	55
MICROCHIP TECHNOLOGY INC	www.microchip.com/graphics	10
MICROSOFT CORPORATION	www.windowsembedded.com/fastforward	4-5
MOUSER ELECTRONICS	www.mouser.com	8
NATIONAL INSTRUMENTS	www.ni.com/imagine	25
NATIONAL SEMICONDUCTOR	www.national.com/switcher	21
NCI	www.nci-usa.com	55
RADICOM RESEARCH INC	www.radi.com	55
RENSESAS TECHNOLOGY CORP	www.america.renesas.com/ReachR8C/c	15
SAELIG CO	www.saelig.com	55
SCIENTIFIC TOOLWORKS	www.scitools.com	22
SEGGER MICROCONTROLLER	www.segger.com	17
SIGNUM SYSTEMS	www.signum.com	55
SMART BEAR SOFTWARE	www.codecollab.com	55
TECH TOOLS	www.tech-tools.com	55
TECHNOLOGIC SYSTEMS	www.embeddedARM.com	24
TERN INC	www.tern.com	55
TEXAS INSTRUMENTS	www.ti.com/ex430-rf	6
THE MATHWORKS	www.mathworks.com	CV2
WIND RIVER	www.windriver.com/clarity	39
XILINX	www.xilinx.com/processor	33

BY Jack G. Ganssle

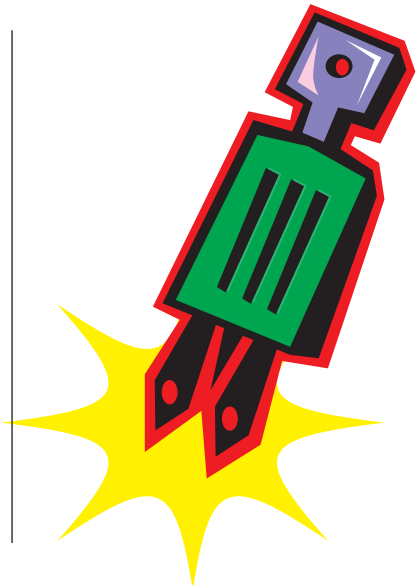
The transistor: sixty years old and still switching

It's hard to say when the electronics age started, but William Sturgeon's 1825 development of the electromagnet laid the seeds that led to Joseph Henry's crude telegraph in 1830, which was the first electrical system used to communicate over long distances (a mile). Just 14 years later, Samuel Morse sent a message by telegraph over a 40-mile link he had strung between Washington DC and Baltimore.

Considering the primitive nature of telegraphy at the time, it's astonishing just how quickly the demand grew. By 1851 Western Union was in business, and in the same decade Cyrus Field had connected the Old and New Worlds via a fragile cable that failed a mere three weeks after the first message was sent. But later attempts succeeded. Instantaneous transatlantic communication quickly lost its novelty.

Although Alexander Graham Bell's 1875 invention of the telephone is universally lauded today, it was a less than practical device till Thomas Edison came up with the carbon microphone two years later. The speaker's voice modulated a pack of carbon granules, changing the circuit's resistance and thus sending a signal to the receiver.

A number of inventors soon came up with the idea of wireless transmission, codified by Guglielmo Marconi's 1896 patent and subsequent demonstrations. Like the telephone and telegraph early radios used neither CPUs, transistors, nor vacuum tubes. Marconi, drawing on the work of others, particularly Nikola Tesla, used a high voltage and spark gap to induce electromagnetic waves into a coil and an antenna. The signals, impossibly noisy by today's standards, radiated all over the spectrum . . . but they worked. In fact, Titanic's famous SOS was broadcast using a 5 KW spark gap set manufactured by the Marconi Wireless Telegraph Company.



Sixty years ago this month, scientists at Bell Labs demonstrated the most important invention of the 20th century: the first real transistor.

The circuits were electrical, not electronic.

Telephone signals, though, degraded quickly over distance while radio remained crude and of limited range. The world desperately needed devices that could control the flow of the newly discovered electron. About this time Ambrose Fleming realized that the strange flow of electricity in a vacuum Edison had stumbled on could rectify an alternating current, which has the happy benefit of detecting radio waves. He invented the first simple vacuum tube diode. But it didn't find much commercial success due to high costs and the current needed by the filament.

In the first decade of the new century, Lee de Forest inserted a grid in the tube between the anode and cathode. With this new control element, a circuit could amplify, oscillate, and switch. Those are the basic operations of any bit of electronics. With the tube, engineers learned they could create radios of fantastic sensitivity, send voices over tens of thousands of miles of cable, and switch ones and zeroes in microseconds. During the four years of World War I, Western Electric alone produced a half million tubes for the U.S. Army. By 1918, over a million a year were being made in the U.S., more than fifty times the pre-war numbers.

Electronics was born.

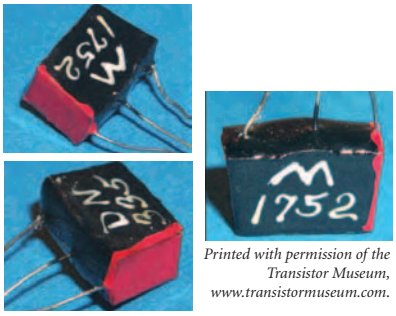
Electronics is defined as "the science dealing with the development and application of devices and systems involving the flow of electrons in a vacuum, in gaseous media, and in

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.

semiconductors," and the word came into being nearly at the same time the tube was created. But that's a lousy definition. I think the difference between electrical and electronic circuits is that the latter uses "active" elements, components that rectify, switch, or amplify. The very first active devices may have been cats whisker crystals, a bit of springy wire touching a raw hunk of galena that works as a primitive diode. I can't find much about their origins, but it seems these crystals first appeared shortly before Fleming did his pioneering vacuum tube research. It's rather ironic that this, the first active element, which predated the tube, was a semiconductor, but that nearly another half century was required to "discover" semiconductors.

Radios originally sported just a few tubes but soon high-end units used dozens. In the late 1960s, I had a military-surplus 1940-era RBC radio receiver that had 19 tubes. Reputedly it cost $2,400 in 1940 (over $33k today). The $600 toilet seat is nothing new.

Increasing capability lead then, as it still does today, to ever-escalating cries for more features, speed, and functionality. The invention of radar in World War II created heavier demands for active electronics. Some sets used hundreds of tubes. Perhaps the crowning achievement of vacuum-tube technology was the ENIAC in 1946, which employed some 18,000. The machine failed every two days. Clearly, the advent of digital technology had pushed tubes to their very limits. A new sort of active element was needed, something that produced less waste heat, used vastly fewer watts, and was reliable.



Printed with permission of the Transistor Museum, www.transistormuseum.com.

Three years later, Bell Labs demonstrated part number M1752, though it was apparently produced only in prototype quantities.

Serendipitously, the very next year Walter Brattain and John Bardeen (who, with William Shockley won the 1956 Nobel Prize for this and related semiconductor work) invented the transistor. Though some claim this was the first "practical" such semiconductor, the Bell Labs scientists had actually constructed a point-contact transistor, a difficult-to-manufacture device that is no longer used and whose use was never widespread.

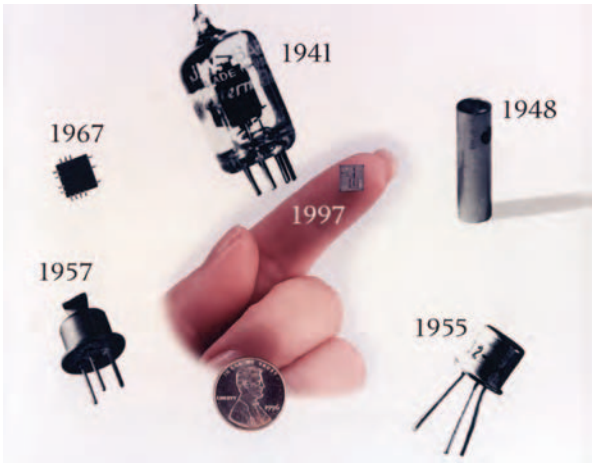
Around 1950 (sources vary), Raytheon produced their CK703, the first commercially available device. At $18 each ($147 in today's inflated dollars), these simply weren't competitive with vacuum tubes, which typically cost around $0.75 each at the time. Though point-contact transistors were tantalizingly close to an ideal active element, something better was needed.

Shockley had continued his semiconductor work, and in 1948 patented the modern junction transistor. Three years later, Bell Labs demonstrated part number M1752 (photos at *http://semiconductormuseum.com/PhotoGallery/PhotoGallery_M1752.htm*), though it was apparently produced only in prototype quantities.

The modern transistor was born. But it didn't immediately revolutionize the electronics industry, which continued its love affair with tubes. It wasn't till 1956 that Japan's ETL Mark 3, probably the first transistorized computer, appeared, but it used 130 point-contact transistors and wasn't a practical, saleable unit. The following year IBM started selling their 608 machine, which used 3,000 germanium transistors. It was the first commercial transistorized computer. The 608 used 90% less power than a comparable machine built using tubes. With a 100 KHz clock, 9 instructions, and 11 msec average multiplication time for two 9-digit BCD numbers, it had 40 words of core memory and weighed 2,400 pounds.

The telephone industry's demand for amplifiers accelerated the development of vacuum tubes, and it unsurprisingly snapped up semiconductor technology. As early as 1952 Bell Telephone installed the first transistorized central office equipment in New Jersey—again, using point contract transistors.

Ma Bell was started by Alexander Graham Bell of course, who started as a teacher of the deaf and who spent much of his career in service to the hearing impaired. So not surprisingly the Bell Corporation waived all patent



"The shape of the transistor has changed dramatically since it was invented at Bell Labs in 1947 as a replacement for the vacuum tube." —Bell Labs, photo from Porticus.org, www.porticus.org/bell/belllabs_transistor.html.

royalties for the very first transistorized consumer product—a hearing aid, around 1953.

Old-timers probably remember Raytheon's CK-722, one of the first commercial junction transistors. It was available in 1953 for about $7 each, a lot of money in those days. I remember buying bags of random transistors from Radio Shack in the '60s that often had CK-722s, probably factory seconds. I have no memory of the cost, but as this was all allowance money it couldn't have been more than a buck or two for a bag of parts.

By late 1955 the same part cost $0.99. Moore's Law didn't exist, but the inexorable collapse of the prices of electronic components had started, entirely enabled by the new semiconductor technology.

Regency Electronics did produce the first commercial transistor radio (eponymously called the TR-1) as early as 1954. (To see videos of this four-transistor radio being assembled check out *http://people.msoe.edu/~reyer/regency/index5.html.*) TI, looking for a market for their new transistors, had approached a number of domestic radio manufacturers but was turned down by all but Regency. A contemporary TI press release about the TR-1 calls the components "*n-p-n* grown junction, germanium triodes." A triode was—and is—a three element vacuum tube.

By the early 1960s, consumers were infatuated with miniature radios (half of the 10 million units sold in 1959 were transistorized). Marketers, then as now anxious to differentiate their products, started using transistor counts to sell product. Although at least one vendor managed to build a radio with just two transistors (schematic here: *www.transistor.org/FAQ/two-transistor.html*), and rarely were more than 8 actually used, often as many as 16 were soldered on the board—most, of course, unconnected. That may be analogous to today's GB wars. How many iPod owners come close to filling their 40 GB drives?

Today, discrete transistors seem almost like anachronisms, although they're still widely used in many demanding applications. Costs range from nearly nothing to tens of dollars or more for certain specialized parts.

Today, discrete transistors seem almost like anachronisms, although they're still widely used . . .

"The first transistor ever assembled, invented in Bell Labs in 1947." Photo and text from Porticus.org, www.porticus.org/bell/belllabs_transistor.html. *(Follow that link to see more historical documents and images about Bell Labs and the transistor.)*

An IC the size of that venerable CK-722 might have hundreds of millions of transistors, each of which costs the buyer a few microcents.

Ironically, some of the problems that plagued vacuum tubes and lead to their near-demise now haunt transistorized products. In 1946, all of the computer capability in the world consumed a few hundred kilowatts. Today a single server farm sucks many megawatts. According to *http://blogs.business2.com/greenwombat/2007/02/photo_originall.html,* in 2005 server farms worldwide needed the equivalent of 14 one-gigawatt power plants. Google's data center in The Dalles, Oregon reputedly has cooling towers four-stories tall.

Transistors come in many varieties, the field-effect transistor (FET) being the most important. Invented in 1960 (drawing on Shockley's work) by John Atalla, it was at first a novelty. RCA introduced a series of logic chips using FETs, but they were used only in specialty, low-power applications due to their low speed. Everyone knew the technology would never replace the much more useful junction transistor.

Now, of course, FETs are the basis of the digital revolution. The speed problems were solved, and their extremely low power requirements made it possible to pack millions on to a single IC.

A three tube radio didn't generate all that much heat, but group 18,000 into a computer and the air conditioning system becomes a significant problem. The same holds true for all kinds of transistors: a single IC with hundreds of millions low-power FETs will thermally self-destruct. So, ironically once again, vendors are grappling with different technologies like multicore to get better MIPs per milliwatt ratios.

At the same time Morse was perfecting the telegraph, the first real electrical system, Rudolf Clausius codified the basic idea of the second law of thermodynamics, which has haunted the entire history of electronics. Multicore may or may not be a solution to MIPs/mW today, but put huge numbers of low-power CPUs on a single core and Clasius's law will surface yet again. I suspect that long before the transistor's 100th birthday entirely novel, low-entropy technologies will be invented. And those, too, will fall to inexorable thermal scaling problems. ■


Mentor KNOWS...
ARM
Development Tools
Application Platform
Operating System
Intellectual Property
Mentor's development tools, operating system, application platform and intellectual property combine together for the world's most complete development solution for ARM-based products.
This month's technology focus: Power Safe File Systems
Join our upcoming WebEx™:
▪ Storing files safely on portable devices
This month's platform focus: Atmel 9261
Download our newest whitepaper:
▪ Storing files safely on portable devices
Development Tools ▪ Operating System ▪ Application Platform ▪ Intellectual Property
Mentor Graphics®
mentor.com/embedded
toll free: 1.800.468.6853 | phone: 251.208.3400
email: embedded_info@mentor.com