What's your Sine? Generate algorithms for DSP

p. 12

>> Programming a 32-bit MCU, p.25
>> 20th Anniversary: Forth data structures, p.35
>> Tear Down: Zune audio player, p.43
>> Ganssle: on people, p.49
POWERED WITH ELECTRICITY, GAS,
AND AUTOMATICALLY-GENERATED CODE.

THAT’S MODEL-BASED DESIGN.

To create a two-mode hybrid powertrain, engineers at GM used models to continuously verify their design, test prototypes, and automatically generate the embedded code. The result: a breakthrough HEV, delivered on time. To learn more, visit mathworks.com/mbd
No INTEGRITY®
No Security

Know INTEGRITY
Know Security

INTEGRITY®: the only secure operating system

www.ghs.com ▲ ph 805.965.6044 ▲ info@ghs.com
Highest performance and lowest active power in one speedy very little package

Another MSP430 breakthrough in ultra-low-power MCU performance

Introducing the new MSP430F5xx generation of MCUs

The first devices in the new 5xx generation of ultra-low-power MCUs extend the MSP430 benefits of high-performance analog integration and ease of design to a new level. The MSP430F54xx devices offer industry-leading active power (as low as 160 µA/MHz) and 1.5-µA standby current (with full RAM and status retention and a 32-bit RTC), all with 25 MHz of peak performance. Intelligent peripherals like the 12-bit ADC draw power only when in operation and an advance power-management module and unified clock system allow the standby and active currents to be dynamically adjusted.

Key benefits
• Leading performance
• Increased design options
• Complete ecosystem

Key applications
• Metering
• Security
• Portable Medical
• Industrial
• General Purpose

Get more information and order samples today at www.ti.com/5xx

The platform bar is a trademark of Texas Instruments. ©2008 TI.
Cover Feature:
What’s your sine? Find the right algorithm for DDFS on a DSP
BY CARLOS ABASCAL
A new approach to direct digital frequency synthesis (DDFS) that combines lookup tables with trigonometric identities lends itself to more efficient implementation on digital signal processors.

Digital ESD:
Practical migration from 8-/16- to 32-bit PIC
BY LUCIO DI JASIO
An insider’s account of moving from Microchip’s 16- to 32-bit PIC MCUs. Note: This article is available on the free digital version of Embedded Systems Magazine. See page 22 for more information.

Compilation strategies for the PIC32
BY JEFFREY O’KEEFE AND MATTHEW LUCKMAN
Unlike conventional compilation that optimizes and generates object code independently for each individual program module, omniscient compilation optimizes based on a view of all modules, across the entire program.

Forth data structures
BY JACK WOEHR
Often what makes Forth seem incomprehensible is not the language itself but a programmer’s poorly conceptualized data structures.
MAKING ONE OF THESE?

...THEN YOU NEED THREADX

ThreadX is Express Logic’s small, fast, royalty-free RTOS that powers over 500 million electronic devices, with millions more produced each month. That’s because ThreadX is easy-to-use, making your development job easier and more likely to finish on time or even ahead of schedule. Developers of consumer, medical, networking, industrial, aerospace, and automotive electronics products rely on ThreadX for their RTOS. Call today to find out how ThreadX can help you bring your next electronic product to market faster. Make it better - make it with ThreadX.

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

expresslogic

Copyright © 2008, Express Logic, Inc.
ThreadX is a registered trademark of Express Logic, Inc. All other trademarks are the property of their respective owners.
Around the ESC world in 40 days

It’s fall and not only is it trade-show travel season—it’s Embedded Systems Conference season. The folks who produce this publication are the same people who organize the ESCs around the world. And as luck would have it (not sure if it’s good luck or bad), six different ESCs are taking place around the world over the course of about six weeks.

The first of those events is actually the first of four ESCs in India. Last year’s event in Bangalore was such a smashing success that we expanded into three other Indian cities: Hyderabad, Pune, and Noida. It was literally standing room only in Bangalore last year, and it appears that this year’s events will be no different.

While ESC India is taking place, ESC UK in Birmingham, England (www.embedded.co.uk/index.php) will be in full swing. The event is really called the Embedded Systems Show (ESS), but the name will change next year as we integrate ESS into the ESC family. Held on October 1 and 2, ESS will have a similar look and feel to ESC, incorporating some of our most popular elements, including the Build Your Own Embedded Systems (BY-OES) program, Live Tear Downs, a host of hands-on classes and workshops, and a jam-packed exhibit floor.

If you’re not familiar with our BY-OES program, you should be. As part of your regular conference fee, you get a complete development kit. This year, the heart of the kit is the popular Beagle Board powered by an ARM-based OMAP3530 microprocessor. Attendees also receive a copy of MontaVista Linux, stored on a 2-Gbyte Numonyx SD Card. During the course of the conference, attendees can participate in a series of classes that are all aimed at configuring the Beagle Board for their specific application. And the best part is that after the conference, attendees get to keep all the hardware and software.

The final stop on the ESC Tour is Boston, which is sizing up to be a mega-event. In addition to the BYOES and Live Tear Downs, we’ve managed to pull in an extremely popular figure for keynote address. He’s known as the father of the cell phone or the inventor of the cell phone. We know him as Dr. Martin Cooper, who really did invent the cell phone while at Motorola. He made that first call on April 3, 1973 to, of all people, the head of research at Bell Labs, Motorola’s biggest competitor at the time.

Unfortunately, there’s no rest for the weary, as the Call for Abstracts for ESC Silicon Valley closes in early October. So, we’ll be sorting through those abstracts, ensuring a great conference in San Jose next April.

Richard Nass is editor in chief of Embedded Systems Design. You may reach him at rnass@techinsights.com.

Richard Nass
ECONOMY MEETS ECOLOGY.

Energy consumption in datacenters is expected to double in the next five years. Yet many businesses still don’t know how much energy their IT is using. So how do you build and manage your IT to reduce energy consumption? With greener software from IBM: a complete range of energy-efficient software to optimize your infrastructure, boost business process efficiency and put practices in place for truly responsible collaboration. With energy at a premium, greener software can help shave millions off your IT and energy budgets. A greener world starts with greener business. Greener business starts with IBM.

SYSTEMS. SOFTWARE. SERVICES. FOR A GREENER WORLD.
Get our green strategy whitepaper at ibm.com/green/software

IBM, the IBM logo and ibm.com are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml. © 2008 IBM Corporation. All rights reserved.
The Newest
Embedded Technologies

New Products from:

- RN-41 Bluetooth® Module
  www.mouser.com/rovingnetworks/a

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

- EZUrIo
  BISM II Bluetooth® Modules
  www.mouser.com/ezurio/a

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 366 leading suppliers.
Perhaps the reason why the chip is so important (Rich Nass, “An insider’s view of the 2008 Embedded Market Study,” September 2008, p.18, www.embedded.com/210200580) is because of the peripheral mix available on a particular chip—for example, the number of counter-timer channels available—could be crucial to somebody off-loading the waveform-generation portion of a BLDC motor-control algorithm onto hardware?

It is interesting how “ecosystem” has been a concern right from the beginning! For example, search for “system concept” or “concept” in this interview with Masatoshi Shima: www.ieee.org/portal/cms_docs_iportals/iportals/aboutus/history_center/oral_history/pdfs/Shima197.pdf

—shirazkaleel
Engineer
Posted in online Forum

Remembering Apollo
I grew up during that time (Jack Ganssle, “Engineering Apollo,” September 2008, p. 53, www.embedded.com/210200582) and followed the missions closely. Looking back, I’m stunned at what was accomplished with such limited resources. Few people today (or then) appreciate the giant leaps that came from the space program, both in technology and in politics. Make no mistake, it was a RACE between the USA and the USSR, and the worldwide political consequences were significant.

Unfortunately, after we “won the race,” we pretty much lost interest in further space exploration.

—daleinaz
HW Engineer
Posted in online Forum

I = SC^2
Embedded systems will be at the forefront of most newer technology (Jack Ganssle, “The real national security issue, www.embedded.com/columns/210001969—the most commented on article on Embedded.com). We need more solar and wind. However, we can also get there with more nuclear. Nuclear is not the infinite solution but it can get us over the hump for the next 50 or so years if we just use it.

—KingMike
Avionics Engineer
Posted in online Forum

As engineers and scientists, we have a special obligation to hold ourselves, those around us, and those who lead us to be ruthlessly unemotional about our evaluation of all of this. Insist on the math and the science. All those who substitute emotion for logic and opinions for facts, regardless of ideological bent, are irresponsible.

—Dave Maples
Posted in online Forum

We welcome your feedback. Letters to the editor may be edited. Send your comments to Richard Nass at rnass@techinsights.com or fill out one of our feedback forms online, under the article you wish to discuss.
The Embedded Systems Conference Boston was created specifically to meet the information needs of the system architects and design engineers who create these complex systems. The technical program at ESC is the Fall’s largest industry conference offering in-depth and unbiased content, that delivers real world solutions from leading technology experts.

Tracks include:

- Commercial & Open Source Operating Systems
- Debugging, Verification & Test
- Design Team Management
- DSP & Multimedia: Algorithms & Implementation
- Hardware Design, including Programmable Logic
- Multi-core Processors & Programming
- Real-time Development
- Security
- Software Development
- Virtualization
- Wired & Wireless Networking

EARLY BIRD REGISTRATION NOW OPEN!
www.cmp-egevents.com/web/escb/registration
Green Engineering
Powered by National Instruments

**MEASURE IT**

- **Acquire** environmental data from thousands of sensors
- **Analyze** power quality and consumption
- **Present** measured data to adhere to regulations

**FIX IT**

- **Design** and model more energy efficient machines
- **Prototype** next-generation energy technologies
- **Deploy** advanced controllers to optimize existing equipment

For more than 30 years, National Instruments has empowered engineers and scientists to measure, diagnose, and solve some of the world’s most complex challenges. Now, through the NI graphical system design platform, engineers and scientists are using modular hardware and flexible software to not only test and measure but also fix inefficient products and processes by rapidly designing, prototyping, and deploying new machines, technologies, and methods. Today, a number of the world’s most pressing issues are being addressed through green engineering applications powered by NI products.

>> Download green engineering resources at ni.com/greenengineering 800 890 1345

©2008 National Instruments. All rights reserved. National Instruments, NI, and ni.com are trademarks of National Instruments. Other product and company names listed are trademarks or trade names of their respective companies. 2008-9759-104-101-D
Perhaps the most fundamental building block of all communication equipment is the sine wave generator. We have ample knowledge of how to generate sine waves using analog electronic hardware. In modern electronics, however, more and more functions once performed by analog circuitry are now performed by digital signal processors or field-programmable gate arrays. Those two types of ICs are the main approaches for developing and implementing signal processing: either the programmable DSP or the ASIC/FPGA.

Lately, high-performance DSPs are using concepts, such as execution pipelining and parallelism of operations, which were unknown to their ancestor the CPU but routine to the FPGA world. DSPs are starting to resemble FPGAs more and more, and although some fundamental differences remain, this hybridization makes it possible to use algorithms originally developed for one environment in the other.

For instance, you can use a very efficient algorithm for computing the sine for an arbitrary phase in a programmable DSP. The method is not new in itself but adapts the original ideas applied to the design of direct digital frequency synthesizer (DDFS) ICs over two decades ago.

It’s important to highlight at this point a crucial difference between algorithms used to compute the sine of a random input phase and those that generate a sine wave. While a sine computation algorithm can be easily converted into a sine wave generator, by providing to its input a monotonically increasing phase sequence, it’s ab-
A new approach to direct digital frequency synthesis uses a combination of lookup tables and trigonometric identities that lends itself to more efficient implementation on digital signal processors.

Find the right **algorithm**
**DDFS** on a **DSP**
Algorithms for the computation and generation of a sine wave in a DSP.

<table>
<thead>
<tr>
<th>Sine computation</th>
<th>Taylor series expansion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lookup table</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sine generation</th>
<th>Recursive trigonometry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resonator</td>
<td></td>
</tr>
</tbody>
</table>

Table 1

<table>
<thead>
<tr>
<th>Number of terms</th>
<th>Taylor Expansion of Sin from $[-\frac{\pi}{2}, \frac{\pi}{2}]$</th>
<th>Taylor Expansion of Sin from $[-\pi, \pi]$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Maximum error</td>
<td>Quantization bits</td>
</tr>
<tr>
<td>3</td>
<td>4.5e-3</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>1.5e-4</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>3.5e-6</td>
<td>18</td>
</tr>
<tr>
<td>6</td>
<td>5.6e-8</td>
<td>24</td>
</tr>
<tr>
<td>7</td>
<td>6.6e-10</td>
<td>30</td>
</tr>
<tr>
<td>8</td>
<td>6.0e-12</td>
<td>&gt;32</td>
</tr>
</tbody>
</table>

Table 2

Absolutely impractical, if not impossible, to use a sine wave generator to calculate the sine of a random phase input.

We'll differentiate these algorithms as sine computation and sine generation with the understanding that a sine computation is a more general method and hence can be easily converted into the latter without significant additional complexity.

In this article, we'll review the most common methods for the computing and generating a sine wave on a programmable DSP, as well as provide a brief introduction to direct digital frequency synthesis (DDFS). We'll then describe a new approach to DDFS based on an algorithm created by combining lookup tables and trigonometric identities. The approach lends itself well to implementation on DSPs such as the Texas Instruments’ C6x.

SINE GENERATION IN A PROGRAMMABLE DSP

Implementing a high-speed sine-wave synthesizer in a programmable DSP is hardly a trivial task; it often requires a careful trade-off between memory and implementation complexity. The problem arises because the standard algorithms used to calculate \( \sin(x) \), as well as any other transcendental function of \( x \), are based on the Taylor series expansion of the function in question, which is a time-consuming procedure. Hence, for a given finite number of CPU cycles, only a limited number of results can be rendered in real time. Several alternative algorithms have been developed to boost the performance of the sine-wave generation. Table 1 presents some of the most popular sine generation and computation algorithms used in a DSP.

**Taylor series:**

The Taylor series expansion of \( \sin(x) \) is shown in Equation 1. Implementing this equation in a high-level language like C is straightforward. One advantage of the method is that the desired precision of the final result can be controlled by choosing the number of terms to include in the calculation, more terms render more precision at the expense of additional computations. The series converges rather fast in the interval \([−\frac{\pi}{2}, \frac{\pi}{2}]\) and remarkably slow from \([−\pi, \pi]\) hence, given the same desired precision, there are two implementation choices: (1) exploiting the symmetry relations in Equation 2 and performing the expansion only in the interval \([−\frac{\pi}{2}, \frac{\pi}{2}]\) with fewer terms, or (2) using more terms (see Table 2) applying the expansion in interval \([−\pi, \pi]\). Since both choices require extra computations, relative advantages should be judged on a case by case basis.

\[
\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + ... \quad (1)
\]

\[
\sin(x) = \sin(x - \frac{\pi}{2}), \quad x \in [\frac{\pi}{2}, \pi]
\]

\[
\sin(x) = -\sin(x - \pi), \quad x \in [\pi, 2\pi] \quad (2)
\]

\[
\cos(x) = \sin(x + \frac{\pi}{2})
\]

Table 2 shows maximum error as a function of the number of terms used in the expansion for each of the intervals and the maximum number of bits that can be used in order to keep the error less than one least significant bit.

**Lookup table:**

The lookup table approach symbolizes the opposite to the Taylor Expansion, in the sense that no calculations are performed in real time. The values of the sine function are precalculated at evenly spaced phase points in the intervals \([0,2\pi]\) and stored in memory during the initialization phase of the algorithm. In this way, a one-to-one mapping is established between each phase value and the address in memory where the table values are stored. The computation of the sine wave in real time entails approximating the input phase to one of the table indexes and retrieving the value stored at that index/address. The phase to index approximation constitutes one of the main sources of error of the algorithm. This error is in the order of the inverse of the table size, which is almost universally chosen to be a power of two.

This algorithm is the fastest sine computation algorithm executing in only one instruction—the memory-read from the index corresponding to the input phase. Its drawback is that the table size becomes impractically large even for modest error goals; for instance an error of the order of \(2^{-12}\) requires a table of 4,096 elements totaling 16 kbytes for 32-bit elements. Obviously this size can be reduced by a factor of four by exploiting the symmetry relations for Sin and Cos.
In 10 years, together with our customers we created the smartphone market, built it up and continue to lead it.

Every day, millions of people use our software in their mobile phones. The 10 000 commercially available applications developed using Symbian OS™ based SDKs enable people around the world to communicate, work and play.

With support for multiple runtimes like .NET, Ruby, Python, Flash, Java and more, we offer the widest choice of tools and development languages. Mobile development starts with Symbian.

It’s no wonder the world’s top five handset vendors use Symbian OS™. Shouldn’t you?

developer.symbian.com
stated in Equation 2 but at the expense of the additional complexity of the algorithm needed to check the phase subinterval prior to lookup table addressing. The amount of overhead associated with this additional complexity more than doubles the execution time of the algorithm, making it a costly proposition. Another way to decrease the table size is to interpolate between the end points of the subinterval where the phase resides; this technique, however, again suffers the aforementioned performance drawbacks.

Recursive iteration:
The recursive iteration algorithm is based in the application of the known trigonometric identities Equation 3 in a recursive manner that is achieved by making \( \alpha[k+1] = \alpha[k] + \Delta \alpha \) and rewriting Equation 3 as in Equation 4. The algorithm is suited only for generation and not computation of the sine function. Another drawback is that, due to the recursive nature of the method, numerical errors accumulate with time producing fluctuations in the output value of the sine wave. A solution to this problem is proposed in John Edward’s article.2

\[
\begin{align*}
\sin(\alpha + \theta) &= \sin(\alpha)\cos(\theta) + \cos(\alpha)\sin(\theta) \\
\cos(\alpha + \theta) &= \cos(\alpha)\cos(\theta) - \sin(\alpha)\sin(\theta)
\end{align*}
\]

(3)

\[
\begin{bmatrix}
\sin(\alpha[k+1]) \\
\cos(\alpha[k+1])
\end{bmatrix} = 
\begin{bmatrix}
\cos(\Delta \alpha) & \sin(\Delta \alpha) \\
-\sin(\Delta \alpha) & \cos(\Delta \alpha)
\end{bmatrix} 
\begin{bmatrix}
\sin(\alpha[k]) \\
\cos(\alpha[k])
\end{bmatrix}
\]

(4)

Resonator:
The resonator implementation takes advantage of the fact that an infinite impulse response (IIR) filter with a pair of complex poles situated in the unit circle generates a constant-amplitude sine wave if excited with an impulse function. The frequency of the oscillation depends on the location of the poles in \( Z \) plane. The transfer function and time domain recursive equation for the IIR filter are shown in Equation 5 along with the IIR block diagram in Figure 1. The IIR has a very efficient implementation in terms of speed and memory usage. Its main drawback is being a sine generation instead of a sine computation algorithm. It also exhibits the same sensitivity to amplitude due to the accumulation of numerical errors with time than the recursive iteration.

\[
H(z) = \frac{b_0}{1 + a_1 z^{-1} + a_2 z^{-2}}
\]

(5)

\[
\begin{align*}
y(k) &= -a_1 y(k-1) - a_2 y(k-2) + b_0 x(k) \\
b_0 &= A \sin(w) \quad a_1 = -2 \cos(w) \quad a_2 = 1 \quad w = \frac{2\pi f}{f_s}
\end{align*}
\]

And initial conditions: \( y(-1) = 0 \quad y(-2) = 0 \quad x(0) = 1 \)

SINE COMPUTATION IN DIRECT DIGITAL FREQUENCY SYNTHESIZERS
Direct digital frequency synthesizer (DDFS) ICs were developed over 20 years ago to digitally generate a sine and/or cosine wave with high frequency resolution and low distortions as a response to the increase demands in the communication field. A popular DDFS architecture originally designed by Tierney consists of a phase accumulator and a phase to amplitude converter as shown in Figure 2.3 At every clock cycle, the phase accumulator is incremented by a programmable quantity \( F_i \), L-1 bits long. The total phase is then used as input to the second block where the sine and/or cosine are calculated. The frequency of the synthesized sine wave \( F_o \) is cal-

![DDFS architecture showing sources of noise.](image1)

![Figure 2](image2)
Rethink small.

Intel's new system-on-a-chip can reduce footprint more than 45%.* Slam dunk.
Put your best footprint forward. Intel enables significant performance/watt improvement for embedded apps by integrating an Intel® architecture CPU, memory and I/O controllers, plus Intel® QuickAssist Technology. Fit more game in your footprint. Go to intel.com/go/rethink

* Compared to previous platform containing the Intel® Pentium® M processor, Intel® 815GME GMCH, Intel® ICH6-M and Intel® IXP465 network processor. Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries. ©2008 Intel Corporation. All rights reserved.
Angle partition for the Sine calculation according to original Sunderland algorithm.

![Figure 3](image)

The phase-to-amplitude converter is the most complex function of the DDFS and has been the target of extensive optimization efforts. In reviewing the methods used in the design of phase-to-amplitude converter of the DDFS, we can expect to find possible candidates for implementation in a DSP. The goal is to find an implementation that could be translated to a programmable DSP, specifically a high-performance TI C6x DSP, and compare its performance with the algorithms previously discussed.

As a reference for comparison, let’s chose the lookup-table algorithm because this was the fastest technique analyzed so far. We can assume that while executing in a single instruction, the algorithm simultaneously computes both the sine and cosine of the input phase, since this function is of widespread use in communications. Both the input and outputs are quantized in 16-bit fixed point format.

Say the size for the uncompressed table is 4,096 32-bit elements (16 kbytes total), with each 32-bit table element containing two contiguous 16-bit groups, the sine and cosine for each of the 4,096 angles from 0 to 4,095/4,096 2π. With the phase quantized in 12 bits the SFDR is, according to Equation 8, approximately 70 dB. To compare performance between algorithms, let’s consider only the number of instructions in the loop kernel, effectively ignoring the effect of the loop’s prologue and epilogue. (See Texas Instruments’ documentation for an in-depth discussion of C6x assembly and optimization.) The loops are fully pipelined and non-interruptible.

Several implementations of phase-to-amplitude converters use techniques similar to those previously discussed for DSP. Langlois,8 for instance, proposed a linear approximation with interpolation in the interval from [0, 2π]. Palomaki and Niitylahti4 have used different methods based on recursive iteration, Taylor and Chebychev series expansions, the last performing at around 70 dB of SFDR for a series with only five terms and CORDIC algorithms, but none of these are suited for DSP implementation due to their iterative nature.

In a seminal paper published in 1984, Sunderland proposed the novel idea of combining lookup tables and trigonometric identities as a means of reducing the table size. The method works as follows: instead of using a single lookup table, the phase is split into several components, and trigonometric identities are used to assemble the desired result. The most straightforward of such identities are precisely Equation 3. Once the input phase has been quantized as an integer between 0 and 4,095, this value is split into two groups of bits of L upper and B lower bits, each representing \( \alpha \in [0, 2^{L-1}] \) and \( \beta \in [0, 1, 2, 3, ..., 2^{B-1}] \) respectively. This corresponds to the graph shown in Figure 3.

The value of \( \sin(\alpha) \), \( \cos(\alpha) \), \( \sin(\beta) \), and \( \cos(\beta) \) are then retrieved from four tables, two tables of size \( 2^{L} \) and two tables of size \( 2^{B} \) elements. These values are used to compute the final sine and cosine using Equation 3.

Sunderland didn’t apply the method exactly as described, but instead to avoid multiplications altogether (a very expen-
Introducing the Perforce Plug-in for Eclipse

Work with Perforce from within your Eclipse-based IDE.

The Perforce Plug-in for Eclipse provides developers with easy access to Perforce SCM from within their Eclipse-based IDE. Functionality includes:

- Quick access to complete file history
- Full support of collaborative development with the ability to merge files
- Ability to work offline when connection to the Perforce Server is unavailable
- Built-in file compare utility and defect tracking support
- Supports the refactoring functionality of the Eclipse environment

The Perforce Plug-in for Eclipse supports both Windows and Linux, and is just one of the many integrations that come with the Perforce SCM System.

Download a free copy of Perforce, no questions asked, from www.perforce.com. Free technical support is available throughout your evaluation.

All trademarks and registered trademarks are property of their respective owners.
sive proposition back in 1984), settled for splitting the phase in three groups and using Equation 9 as an approximation of Equation 3.

\[
\sin(A + B + C) \approx \sin(A + B) + \cos(A)\sin(C)
\]  

Equation 9

At first glance, the original idea suggested by Sunderland’s technique seems to lend itself well for implementation in a programmable DSP. It succinctly proposes a reasonable trade-off: a sizeable decrease in the lookup table size at the expense of a modest increase in the number of calculations. The following reasons clarify why Sunderland’s original works well with the specific TI C6x DSP family architecture.

1. Sunderland’s original requires, according to Equation 3, two multiplications and two additions per every output value: a C6x DSP has a total of eight execution units, among them two adders and two multipliers, hence it’s not a stretch to expect that all operations could be performed in a single clock cycle.

2. The lookup table has a small footprint: in order to achieve maximum throughput, the lookup table must be stored in (precious) internal DSP memory, hence small size is highly desired.

3. Conditional instructions are absent: per every input-phase value, the same operations are executed in a serial fashion. In other words, memory lookup is followed by the operations corresponding to Equation 3, which is very well

**Listing 1** C and assembly code of the sin/cos computation algorithm.

```c
void GetCosSin(Int16 *restrict angle, Int16 *restrict y, Int32 size) {
    register Int32 Alfa, Beta, Angle, i;
    register Int32 si_co_Alfa, si_co_Beta;

    #pragma MUST_ITERATE(16,1024,16);
    for(i=0; i<size; i++){
        Angle = (Int32)*angle++;
        Alfa = _extu(Angle,(32-2*SinCos_TABLE_BITS),(32-SinCos_TABLE_BITS));
        Beta = _extu(Angle,(32-1*SinCos_TABLE_BITS),(32-SinCos_TABLE_BITS));
        si_co_Alfa = Alfa_Sin_Cos[Alfa];
        si_co_Beta = Beta_Sin_Cos[Beta];
        //                  h  l          h  l
        *y++ = _ssub(_smpy(si_co_Alfa, si_co_Beta),
                      _smpyh(si_co_Alfa, si_co_Beta))>>16;
        *y++ = _sadd(_smpyhl(si_co_Alfa, si_co_Beta),
                      _smpylh(si_co_Alfa, si_co_Beta))>>16;
    }
}
```

**$CSL2:** ; PIPED LOOP KERNEL

```
             SHR .S1 A3,16,A7 ; |96| <0,16>
             SMPYHL .M2X B8,A8,B9 ; |98| <1,13>
             BDEC .S2 $CSL2,B0 ; |84| <1,13>
             SMPY .M1X A8,B8,A3 ; |96| <1,13>
             LDW .D1T1 *+A5[A7],A8 ; |86| <3,7>
             LDW .D2T2 *+B6[B5],B8 ; |91| <3,7>
             SUB .L1 A1,1,A1 ; <0,17>
             SHR .S2 B4,16,B5 ; |98| <0,17>
             STH .D2T1 A7,"++B7(4) ; |96| <0,17>
             EXTU .S1 A3,20,26,A7 ; |86| <4,5>
             MV .L2X A3,B4 ; |86| <4,5>
```

*dwpsnfile “NCO_16.c”, line 99, column 0, is_stmt

```
             SUB .S1 A0,1,A0 ; <0,18>
             STH .D2T2 B5,"++B7(2) ; |98| <0,18>
             SADD .L2 B16,B9,B4 ; |98| <1,15>
             SSUB .L1 A3,A4,A3 ; |96| <1,15>
             SMPYHL .M2X A8,B8,B16 ; |98| <2,12>
             SMPYH .M1X A8,B8,A4 ; |96| <2,12>
             EXTU .S2 B4,26,26,B5 ; |91| <4,6>
             LDH .D1T1 *A6++,A3 ; |86| <6,0>
```
suited for C6x architecture since any conditional operation inside a pipelined loop will force it to be at least six instructions long (see TI’s TMS320C6000 Programmer’s Guide\(^9\)).

At this point, we should analyze the implementation of this algorithm for 16 signed short integer format in both input and outputs. The C code for the algorithm is shown in Listing 1 with the corresponding generated assembly code. Note that the loop executes in three instructions compared with one for the lookup table. Furthermore, if the loop is unrolled by a factor of two, it will execute in five instructions generating two pairs of sine/cosine outputs per iteration. This level of performance compares very well with the lookup table implementation; although still 2.5 times slower, it uses 1/32 of the memory. Note as well that while the execution speed remains constant with the table sizes, the saving in memory use continues to growth at a rate of \(2^{n-1}\). This algorithm can be easily modified to operate as a sine/cosine generator. As a generator, there is no need to input an array with phase values; just input the constant phase increment \(\Delta f\) and let a persistent memory variable maintain the phase between calls. A 32 bits INT should suffice even for the most demanding frequency-precision requirements. In this type of application, Equation 7 must be taken into account.

One source of error in this implementation is the truncation of the least significant bit in the calculation of Equation 3. However, since, for the chosen table size, this error is relatively small compared with the error in the phase approximation, I ignored the error. If the application calls for an SFDR such that the uncompressed table size needs to be more than 14 bits, the truncation error must then be considered and it might be necessary to perform rounding of the results at the expense of lower throughput.

Note that the same kind of performance (three cycles/loop without unrolling and five cycles/loop with unroll of two), can be obtained for a floating-point implementation of the same algorithm, using a C67x DSP. Although floating-point DSPs are slower than fixed-point DSPs, in a floating-point unit all operations and suffers from accumulative truncation errors in the fixed-point implementation that degrade its performance. For these reasons, it’s only practical in floating-point implementations where design constraints (such as an SFDR) call for the use of huge table sizes.

The algorithm has been successfully deployed in the field. Finally, some of the initialization functions used in the algorithm have been omitted due to space constraints. For those interested in the full version of the algorithm or in either the floating-point or the 32-bit fixed-point versions, please send an email to carlos.abascal@harris.com.

Carlos Abascal is signal processing and communications engineer in the Government Communications Division of Harris Corp. He has extensive experience in embedded microprocessor and DSP software development and has expertise in the theory and the implementation of linear and nonlinear adaptive predistortion for RF power amplifiers, digital modems, and software-defined radios in DSPs. He may be contacted at carlos.abascal@harris.com.

ENDNOTES:

An insider’s account of moving from Microchip’s 16- to 32-bit PIC MCUs

Practical migration from 8-/16- to 32-bit PIC

BY LUCIO DI JASIO

It was less then a month after my first book on programming 16-bit PIC24 microcontrollers had been published, when I heard through the Microchip grapevine that a new 32-bit PIC32 microcontroller (MCU) had just come out of the “ovens” and a few alpha samples were already working on the benches. The rumor mill said it was based on a MIPS core, but at the same time there were claims of compatibility with the 16-bit pin out and the peripheral set of the PIC24 family. It was simply too much for me to ignore. Off I went to get one of those early samples and a fresh beta copy of the GNU-based MPLAB C32 C compiler.

I simply had to see for myself what this new product looked like. Would it still feel like a PIC MCU? Would it work on the same demo boards? After all, I had just written 15 chapters worth of 16-bit code and examples in C for the PIC24. To make a long story short, less than one month later, not only was I finished porting the code, but I was already working on a new book based on my experiences with the PIC32!

The following is a brief recount of what happened during that month. I’d love to be able to start by saying that I followed the best design rules and that I read the datasheet first, from cover to cover, but I would be lying! I did exactly what you would have done. I launched the MPLAB integrated development environment, which loaded my last PIC24 project, and hit the F10 button to try and build it right away.
MEN Micro Cuts Development Time and Costs with New ESMexpress® Ruggedized COM Standard

MEN Micro moves embedded technology forward again...this time by enabling a significant reduction in the time and costs of developing computer-on-modules (COMs) for rugged, harsh and mission-critical environments. It's done with ESMexpress®, a new computing standard in development to be the ANSI-VITA 59 (RSE Rugged System-On-Module Express) Standard.

COMs are complete computers on a mezzanine board that use a standard CPU with I/O configuration only required on a carrier board to allow for individual functionality tailored to the specific application. ESMexpress combines this model with advanced cooling technologies, the latest serial buses and rugged components to ensure safe, reliable operation in harsh environments found in areas as diverse as the railway, avionics, industrial automation, medical engineering and mobile industries.

The XM1, one of the first available ESMexpress products from MEN Micro, features the first-generation Intel® Atom™ processor (Z530 at 1.6 GHz or Z510 at 1.1 GHz) based on 45 nm technology coupled with 1 GB of soldered DDR2 SDRAM for significantly lower power dissipation, reduction in design costs and space-saving design flexibility.

Although the ESMexpress-based XM1 dissipates an impressive 7 Watts of power using the standard's advanced fanless cooling system, ESMexpress itself enables power dissipation of up to 35 Watts while providing added EMC protection by mounting the populated PCB to a frame and completely enclosing the module in a hermetically-sealed aluminum housing. The high pressure caused by the screw joints between the housing and the PCB facilitates the thermal connection of the components. If additional cooling is needed, the housing is either connected to an external heat dissipation system (conduction) or combined with a heat sink for heat dissipation (convection).

ESMexpress provides extreme resistance against shock and vibration. Eight screws secure the module to the carrier board. In addition, a mechanically-robust connector specified for MIL and railway applications supports differential signals with up to 8 GHz (16 Gbps), features a stacking height of 5 mm with a minimum tolerance of +/-0.3 mm, is equipped with fixed contacts for power supply and is specified for an operating temperature of -55°C to +125°C. The Intel-based XM1 offers a screened temperature range of -40°C to +85°C.

The electrical signals, distributed on two 120-pin connectors, are only defined for modern serial buses. For PCI Express there are four single lane ports (4 x1) and one port that can be configured as 1 x16, 1 x8, 2 x4 or 2 x1. Other ports include three 1 Gigabit Ethernet (also as 10 Gigabit), eight USB, three SATA, SDVO, LVDS, HD Audio ports as well as several utility signals and a single 12 V power supply. A fixed pin assignment guarantees that ESMexpress modules remain interchangeable.

Taking embedded technology for harsh environments to new levels of efficiency, performance and reliability is what MEN Micro is all about. Our new ESMexpress System-on-Module Standard and our XM1 are our latest examples of that. There's plenty more to come!
Twenty years in the making, the Embedded Systems Conferences have surpassed all others to become the largest global brand to connect electronics engineers and vendor companies in three major markets – North America, China, Taiwan and now India. Widely acclaimed as the gold standard in embedded training and technical development, the Embedded Systems Conference India will provide high-quality content, delivered by the industry’s most experienced and respected experts.

Roadshow:
- Hyderabad: September 25-26, 2008
- Pune: September 29-30, 2008
- Nodia: October 7-8, 2008

For exhibiting opportunities please contact

Sean Raman
Event Sales Director, TechInsights
SRAMAN@TECHINSIGHTS.COM
Phone: 415-947-6622

For attendee information, please visit our event website: www.embedded.com/esc/india
Phase 2 of PIC32 contest

It’s like PBS’s show Everyday Edisons (www.everydayedisons.com), except after you pitch your idea, you have to design the product yourself: spec it, build it, program it, and test it—all in your spare time. Oh, and you have to use Microchip’s PIC32 and can’t spend more than $400 on extra components.

The PIC32 Design Challenge, sponsored by Microchip, DigiKey, and TechInsights (publishers of this magazine), is a year-long design contest to get engineers to share all those creative ideas, using the PIC32, while hopefully having some fun. Broken into four phases, the contest rewards participants with the kind of “toys” that make the home tinkerer salivate: Phase 1 winners (all 128 of them) get a PIC32 starter kit, PIC32 I/O expansion board, prototyping board, and a webcam; the 32 Phase 2 survivors earn a MPLAB C32 C Compiler, MPLAB Real Ice, and MPLAB Real Ice Trace Kit; the eight Phase 3 winners receive a Intronix 34 Channel LA1034 Logicport Logic Analyzer; and the first place Phase 4 winner gets the home theater system.

Now in Phase 2, the hardware stage, the 128 contestants have received their PIC32 kits and are competing for 32 slots in Phase 3. On October 29, the judges will announce the Phase 3 contestants.

The judges evaluate designs for creativity, efficiency, usefulness, and technical complexity. In addition to the three appointed judges, the community makes up the fourth judge.


—Richard Nass, Editor in chief and contest judge

DON’T FORGET TO VOTE.

No waiting in long lines to cast your vote. You can easily comment and rate designs online by signing up as a community member.

ENTRY HIGHLIGHTS:

Carb Counter Plus — The Diabetic and Diet Assistant
Inspired by his son, this engineer wants to help people stay healthy.
http://mypic32.com/web/guest/contestantsprofiles?profileID=13642

Network Enabled Access Control Terminal (v2.0)
Keep unwanted visitors out.
http://mypic32.com/web/guest/contestantsprofiles?profileID=14278

The Wand: 3D pen with force feedback
A human-interface device for use in 3D software.
http://mypic32.com/web/guest/contestantsprofiles?profileID=24965

PIC’n The Beehive
Fight Colony Collapse Disorder.
http://mypic32.com/web/guest/contestantsprofiles?profileID=40796

PIC32 SunBot
This robot studies the sun.
http://mypic32.com/web/guest/contestantsprofiles?profileID=13961

Lawn Mapping Aqua-Save Sprinkler
For responsible lawn watering.
http://mypic32.com/web/guest/contestantsprofiles?profileID=40817

Audial Illuminator
Helps hearing-impaired users take action when a signal light flashes.
http://mypic32.com/web/guest/contestantsprofiles?profileID=13961
A long list of errors filled the output window. Much to my surprise, all of the errors reported were apparently related only to my usage of the binary notation (0b00000000), a non-standard extension of the C language. I was trying to compile the first code example from Chapter 3 of the 16-bit book. This is a very simple piece of code that’s supposed to illustrate the command of I/Os, accurate timing, and flow control (for loops) in C. I quickly decided to convert all the literals to the standard hexadecimal notation (0x00) and voila! The compiler and linker parsed my code with no errors.

I powered the board and watched it incredulously for a few seconds—no smoke!

I mean, something was clearly happening on the board, but it didn’t resemble even remotely what I was expecting. Let me explain. In the first three chapters of the 16-bit book, I show readers how to produce a first “Hello world” kind of example in C. In those chapters, I claim that the traditional way of doing it, sending a string of characters to a terminal, is not realistic or appropriate for the embedded-control world of applications. Instead, I take a more “entertaining” approach by making a single row of eight LEDs flash rhythmically so that, when the board is held in hand and waved, it “paints” the desired message, thanks to the human eye’s natural image persistence. This is actually easier to code than to describe.

**A DIFFERENT CLOCK**

Fact is, the PIC32 seemed to get both the I/O pins and the timing all wrong.

As is customary, it was only at this point that I cracked open the datasheet and started working my way back to the root of the problem(s). It turns out that the PIC32’s clock-generation module is a bit more complex than that of the PIC24F I had used in the 16-bit book. In fact, the PIC32’s module more closely resembles the oscillator modules available on the latest PIC24H 16-bit MCU families. Also, in the PIC32 architecture, most peripheral modules are tied to a separate peripheral bus that can operate at a different frequency—lower than the system clock—to help manage power and, of course, EMI.

With a little patience, I figured out how to configure the peripheral bus to operate at the same frequency that the PIC24F used in the same project (16 MHz peripheral bus). I figured also that the same number of instructions could now be executed using only half the system frequency required by the PIC24F, since the PIC32 core can execute one instruction every clock cycle.

**JTAG DEFAULTS TO ON**

After clearing the clock issue, I rapidly glanced at the timer modules. There were five of them. They looked absolutely identical to the PIC24F and, further back in PIC MCU history, compatible all the way back to the PIC16C74 (circa 1994). I continued on to verify the I/O ports: same structures, same number of pins, same “historical” register names, a compatibility trail that can be extended perhaps all the way back to the first PIC16C54 (circa 1991).

I did one last quick check of the A/D converter module, which is a diffi-
cult peripheral for most PIC MCU beginners to learn. It multiplexes its inputs on top of an I/O port (PORTB on most 16-bit PIC devices), and it takes precedence at power up, so it won’t let your digital inputs work unless you configure it properly. It was remarkably compatible with the PIC24, and I still couldn’t explain the strange behavior of the LEDs.

Looking more closely, I realized that there were four LEDs, in particular, that either never lit up or were, apparently, constantly on. So, I went back to the datasheet once more to check them out in the pin-out diagrams, and there I found the culprit: the JTAG port.

While the PIC32, like most PIC MCUs, offers a convenient but proprietary two-wire serial interface for programming and debugging (called the In-Circuit Serial Programming interface), the PIC32 also sports an industry-standard, four-wire (E)JTAG port that not only allows boundary scans, but also enables complete device programming and debugging control. This is expected in the rarefied atmosphere of the high pin-count, 32-bit world; consequently, the PIC32 makes both interfaces active by default at power up. If not required, it’s up to the application program to disable the JTAG port in order to reclaim some of PORTA I/Os.

Once I took care of the JTAG port, my first PIC32 project started working as expected, and sent its first “Hello” to the world, shown in Figure 1.

The simple lessons learned thus far (oscillator configuration and JTAG port) quickly proved to be the key to the compatibility of most other projects in the early chapters of the 16-bit book, which ported quite uneventfully in the following few days of “exploration.” I used the UARTs to communicate with a PC. I used one of the SPI interfaces to communicate with a serial EEPROM. I used the Parallel Master Port to communicate with an LCD display module. I used the A/D to read a potentiometer first and a temperature sensor later, demonstrating how to interface the PIC32 with the analog world. All of these modules were working exactly how I was expecting them to and, apart from some extensions to the modules’ capabilities, I found that my 16-bit code kept working with hardly any changes required.

Table 1 shows a PIC24F AD1CON side-by-side register comparison.

### MEASURING PERFORMANCE

In those early days, my curiosity was consumed by the promise of performance that the PIC32 brought. In chapter 4 of the 16-bit book, “Numb3rs,” I had been counting the instruction cycles required to perform basic arithmetic op-

---

**Table 1**

<table>
<thead>
<tr>
<th>PIC24F AD1CON Register:</th>
<th>PIC32MX AD1CON Register:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Upper Byte</strong></td>
<td><strong>Upper Byte</strong></td>
</tr>
<tr>
<td>R/W:0</td>
<td>R/W:0</td>
</tr>
<tr>
<td>R/W:0</td>
<td>R/W:0</td>
</tr>
<tr>
<td>R/W:0</td>
<td>R/W:0</td>
</tr>
<tr>
<td>R/W:0</td>
<td>R/W:0</td>
</tr>
<tr>
<td>R/W:0</td>
<td>R/W:0</td>
</tr>
<tr>
<td>ADON</td>
<td>ON</td>
</tr>
<tr>
<td>ADSIDL</td>
<td>FRZ</td>
</tr>
<tr>
<td>bit 15</td>
<td>bit 15</td>
</tr>
<tr>
<td><strong>Lower Byte</strong></td>
<td><strong>Lower Byte</strong></td>
</tr>
<tr>
<td>R/W:0</td>
<td>R/W:0</td>
</tr>
<tr>
<td>R/W:0</td>
<td>R/W:0</td>
</tr>
<tr>
<td>R/W:0</td>
<td>R/W:0</td>
</tr>
<tr>
<td>R/W:0</td>
<td>R/W:0</td>
</tr>
<tr>
<td>R/W:0</td>
<td>R/W:0</td>
</tr>
<tr>
<td>SSRC2</td>
<td>SSRC2</td>
</tr>
<tr>
<td>SSRC1</td>
<td>SSRC1</td>
</tr>
<tr>
<td>SSRC0</td>
<td>SSRC0</td>
</tr>
<tr>
<td>bit 7</td>
<td>bit 7</td>
</tr>
<tr>
<td>HCS</td>
<td>HCS</td>
</tr>
<tr>
<td>ASAM</td>
<td>SAMP</td>
</tr>
<tr>
<td>DONE</td>
<td>DONE</td>
</tr>
<tr>
<td>bit 0</td>
<td>bit 0</td>
</tr>
</tbody>
</table>
erations and comparing them with various integer and floating-point types. This was a rational way to proceed in a world where clock cycles corresponded exactly to instructions executed, as happens in the core of the PIC24 and dsPIC DSC. But the PIC32 core, being of MIPS descent, added a new twist to the game. The number of instructions executed per clock cycle can vary, as wait states can be inserted when executing code faster than the Flash memory’s rated speed (just add one clock cycle every 30 MHz); or can be removed thanks to a prefetch mechanism (capable of fetching four instructions at once). Finally, a cache memory can be activated, further improving the performance at high speed.

The PIC32’s cache memory made the count of cycles somewhat unpredictable and perhaps meaningless. I felt like I had just upgraded from a stock muscle car to a Formula One race car. So, I decided I needed to include a new chapter in the 32-bit book about performance “tuning” of the PIC32. To give the PIC32 a serious workload, I rescued an old code example from my university days, when I was learning about basic digital signal processing: a Fast Fourier Transform. I used standard floating-point arithmetic with no hand or compiler optimizations. I also used a 32-bit timer to allow the PIC32 to time its own work, and then I gradually started playing with the new performance options.

First, I enabled the instruction prefetch, then I turned up the cache memory, then I manually trimmed the wait states. The performance gains were dramatic at first and more gradual later, as I kept refining the configuration. Eventually, I realized that the optimal configuration would have been application specific, but a good starting point was offered by a function SYSTEM-ConfigPerformance(), part of the standard device libraries.

**LEARNING THE LIBRARIES**

This was my first encounter with the “standard” peripheral libraries, and the beginning of a long love/hate relationship of sorts. Having spent many years of code development in assembly on very small 8-bit devices with performance levels that required almost constant use of hand-optimized/custom code, I had been working mostly on my own; eventually developing my own set of libraries.

This time, more than one year before the release to production of the PIC32, not only had the 16-bit libraries already been ported, but they were extended to support a number of new features. I had no more excuses—I had to get in line and learn how to use them myself. See Listings 1 and 2 for a code example using the Peripheral Libraries.

By using the new libraries, the code compatibility between 16-bit and 32-bit applications becomes “absolute.” Even the smallest difference at the peripheral register level can be removed completely from the application code. In fact, this allows a single application to run on either a 16- or 32-bit MCU, and a developer can target both architectures while maintaining a unified codebase.

But, while the hardware-control register names are spelled (and each one of their bits is detailed) in the device datasheet, all of the function/macro names and their parameters are not. Most of the time, I found myself having to compare the individual include files with the device datasheet, trying to guess which combination of control bits would correspond to a specific library parameter. This was especially annoying with the simplest (thinnest) of the libraries, such as the I/O port manipulation, where the benefit of the library abstraction layer was more questionable to me.

Eventually, I found a balanced compromise could be taken; I would use the traditional method of accessing the most basic peripherals (I/O ports and timers, for example), but I would use the libraries when more complex/new peripherals were to be used. So, I quickly moved through several chapters of code, making practically no modifications whatsoever. These chapters included: SD/MMC interfacing, FAT16 file I/O and even WAV music file playback.

The usefulness of the libraries became really apparent when I decided to dig deeper into the subject of interrupts, first, and later when I started us-

---

**Listing 1** Example of direct PORT manipulation.

```c
LATA = 0;  // clear PORTA output latch
TRISA = 0xFF3 ;  // configure PORTA pin 2 and 3 output
```

**Listing 2** Example of PORT manipulation using the peripheral libraries.

```c
mPORTAWrite( 0);
PORTSetPinsDigitalOut(IOPORT_A, IOPORT_PIN_2 | IOPORT_PIN_3);
```
ing the new Direct Memory Access (DMA) module of the PIC32.

INTERRUPTS AND DETERMINISM
The PIC32 offers two options to manage interrupts: a single vectored mode, which is more similar to the way the PIC16/18 8-bit architectures operate (incidentally more RTOS friendly, too), and a multivectored mode that closely resembles the way the 16-bit PIC24 MCUs and dsPIC DSCs operate. Setting things up one way or the other was a breeze with the interrupt.h library.

It was when I tried to port the code in Chapter 12: “The Dark Screen” that things got really interesting. With the PIC24, I had been able to demonstrate how a simple SPI port, three resistors, a couple of interrupts, and a little creativity could go a long way to generate a composite video signal and practically transform any TV set into a monochrome display. Creating the video signal required very precise coordination between the interrupt code and peripherals. In practice, since even a single clock-cycle difference on the output timing could result in a visible jitter on the left edge of the screen (making all vertical lines jagged), the exercise turned out to be an ideal magnifying lens for interrupt “determinism,” a characteristic where PIC architectures traditionally shine. Unfortunately, the instruction pre-fetch and memory cache mechanisms are, by definition, “nondeterministic.”

After much head scratching, it finally dawned on me. I was trying to fit a round peg in a square hole. A 32-bit core is designed for performance. Its mission is to run through C code as fast and efficiently as possible, leaving any hard timing work to its peripherals. The DMA peripheral, in particular, was a much better tool for the job.

Eventually, I figured out how to synchronize DMA data transfers to the SPI port, with a timer directly producing the composite video signal. The new solution offered deterministic timing, but also reduced the CPU overhead from about 25% to less than 5%. After a few hours of work, I had compelling 2D and 3D video demos up and running, together with animation and higher-resolution displays all the way up to a monochrome VGA (see Figures 2 and 3 for examples).

Exploring the PIC32 quickly became a very addictive activity, and the results were so rewarding that I ended up embarking on a new 32-bit book project. The compatibility of the PIC32 with the previous generation of 16-bit PIC24 microcontrollers turned out to be quite seamless. The speed and performance of the new MIPS core did not fail to impress me, expanding considerably the envelope of applications beyond what any previous PIC MCU could. But, perhaps most interestingly, the new PIC32 still “felt” like a PIC MCU and behaved like a PIC MCU.

Lucio Di Jasio joined Microchip Technology Inc. in 1995 as a field application engineer. Currently, he is a PIC32 marketing manager in Microchip’s High Performance Microcontroller Division. Di Jasio is also the author of two books: Programming 16-bit Microcontrollers in C: Learning to Fly the PIC24 and Programming 32-bit Microcontrollers in C: Exploring the PIC32, both published by Elsevier. He received his MSEE from the University of Trieste, Italy in 1990. You may reach him at lucio@dijasio@microchip.com.

After much head scratching, it finally dawned on me. I was trying to fit a round peg in a square hole. A 32-bit core is designed for performance.
Hammer Down Your Power Consumption with picoPower™!

Performance and power consumption have always been key elements in the development of AVR® microcontrollers. Today’s increasing use of battery and signal line powered applications makes power consumption criteria more important than ever. To meet the tough requirements of modern microcontrollers, Atmel® has combined more than ten years of low power research and development into picoPower technology.

picoPower enables tinyAVR®, megaAVR® and XMEGA™ microcontrollers to achieve the industry’s lowest power consumption. Why be satisfied with microamps when you can have nanoamps? With Atmel MCUs today’s embedded designers get systems using a mere 650 nA running a real-time clock (RTC) and only 100 nA in sleep mode. Combined with several other innovative techniques, picoPower microcontrollers help you reduce your applications power consumption without compromising system performance!

Visit our website to learn how picoPower can help you hammer down the power consumption of your next designs. PLUS, get a chance to apply for a free AVR design kit!

THE Performance Choice for Lowest-Power Microcontrollers

http://www.atmel.com/picopower/
The PIC32 is a major departure from Microchip’s bread-and-butter offering of 8- and 16-bit microcontrollers, so developing code for the PIC32 poses a new set of challenges. The highest non-volatile-memory (NVM) density on 8-/16-bit PIC MCUs is 128 kbytes, SRAM is limited to 4 kbytes or less, and 16 registers is the maximum. Midrange PICs have as few as one or two registers and even smaller memory densities.

All 8-/16-bit PIC MCUs have banked memories that require assembly language or nonstandard C extensions to address efficiently, resulting in nonportable code. Thus, in spite of the common peripheral set and development environment shared by the PIC32 and its 8-/16-bit precursors, migrating nonstandard legacy code could pose serious complications.

In contrast to previous PICs, PIC32 devices have as much as 512 kbytes of flash, up to 32 kbytes of SRAM memory, and 32 general purpose registers. Also unlike 8-bit PICs, the PIC32 memory space is linear, so no complicated addressing schemes are required.

Unlike conventional compilation that optimizes and generates object code independently for each individual program module, omniscient compilation optimizes based on a view of all modules, across the entire program.
With fast DSP instructions for multiply and divide, a 256-byte instruction cache, 5-stage pipeline, direct memory access, and fast context switching, the PIC32 offers instruction throughput of 1.56 DMIPS (Dhrystone million instructions per second)/MHz—the highest of any microcontroller in its class. Its 80 MHz maximum clock provides PIC32 users with unparalleled throughput and flexibility. So, the question is how does an engineer develop code to take full advantage of the PIC32’s horsepower?

One way to squeeze the maximum performance from a PIC32 is to choose a compilation method that exploits the benefits of the architecture while offering a seamless up/down migration path for legacy code. Basically, there are two approaches to compilation for the PIC32 architecture: conventional compilation that optimizes and generates object code independently for each individual program module and “omniscient” compilation that optimizes code based on a view of all program modules, across the entire program.

**PITFALLS OF CONVENTIONAL COMPILATION**

Conventional compilation technology (shown in Figure 1) shadows the modular embedded software design process, in which programs are broken up into modules—partly to accommodate their increasing complexity and partly to distribute programming tasks among teams of engineers to speed up the development process.

Compilers generate code in the same way, individually compiling each module into an independent sequence of low-level machine instructions, without any knowledge about what is in the other modules. Once all the modules are compiled, a linker links the modules together, along with any code being used from precompiled libraries.

The drawback of this approach is that the compiler never has complete information about the program being compiled. The “global” optimization claimed by many vendors is done only within single modules. There is no optimization across all program modules, which leads to the suboptimal allocation of the stack, registers, and memories.

Conventional compilers have restrictive, fixed calling conventions specified by core and MCU vendors to accommodate a traditional compiler’s “ignorance” of the whole program. Overwriting the data in a single register can have catastrophic consequences. Consider the spectacular crash of the European Space Agency’s Arianne rocket, which directed itself toward the Earth, rather than away from it because of a data overflow into the wrong register.

In conventional compilation technology, calling conventions are used to restrict the use of some registers specifically to avoid this type of error. Another thing conventional compilers do to prevent overwriting the registers is to save and restore very large contexts, during interrupts. Since the compiler has no way of knowing when a register is available or what data from another module might be written to it, the only way to ensure registers are not overwritten is to reserve specific registers only for some kinds of data (such as function parameters) and to always save all registers possibly used to the stack when calling a function or saving a context during an interrupt.

These calling strategies attempt to balance which registers are allocated for function parameters and which registers can be used by the called function for internal calculations, based on “average” code. If too many registers are used for parameters, there may not be enough available for the function’s code. If too few are used for function parameters, the stack may be overutilized, wasting both cycles and SRAM resources.

In the case of interrupts, again, a conventional compiler has no way of knowing which registers are used by the interrupt code. Thus, in order to prevent memory overwrites, most compilers save every register that might be used by an interrupt. This is the safest
Code Review

Learn how with this FREE book
Ten essays about code review including the largest case study of lightweight peer code review. www.codereviewbook.com

Code Collaborator
Tool-assisted peer code review with Code Collaborator means no more busy-work, marked-up print-outs, or meetings. Start a free trial today and cut up to 75% of the time out of code review.

www.CodeCollaborator.com

Time-Saving Automation
Package file changes, send e-mail notifications, and collect review metrics automatically.

Threaded Conversations
Defects, comments, and version control files stay linked together.

Communication Options
Real-time chat for instant communication. E-mail and RSS notifications for posted comments.

Best Kept Secrets of Peer Code Review
Modern Approaches Practical Advice

2008 Jolt Winner
Collaboration Tools

SmartBear SOFTWARE
approach. However, since the number of cycles used is a direct function of the number of registers that are saved and restored, interrupt latency can be longer than necessary and performance can suffer. Cycles spent saving and restoring more registers than required for the context are basically wasted.

Another aspect of writing code for the PIC32 is the existence of legacy code. The C language assumes a linear address space, and traditional compilers also assume a linear address space. They have no way of knowing which objects are stored in which memory locations.

When the MCU’s address space is linear, as is the case with the PIC32, there is no problem. When there are separate memory banks or regions, as in other PIC architectures, the software engineer must specify the addresses where objects are stored with assembly language or C extensions, making much legacy PIC code nonportable to the newer devices.

OMNISCIENT CODE GENERATION

Newer compilers are now available with omniscient code generation (OCG) technology. A compiler with OCG can eliminate arbitrary restrictions on register usage and save only those context registers that are used for each particular interrupt. OCG works by collecting comprehensive data on register, stack, pointer, object, and variable declarations from all program modules before compiling the code (shown in Figure 2).

The OCG compiler analyzes all the program modules in one step and extracts a call graph structure shown Figure 3. Based on the call graph, the OCG compiler creates a pointer reference graph that tracks each instance of a variable having its address taken, plus each instance of an assignment of one pointer to another (either directly, via function return, function parameter passing, or indirectly via another pointer).

The compiler then identifies all objects that can possibly be referenced by each pointer. This information is used to determine exactly the size and scope for each pointer variable (shown in Figure 4). With the PIC32 devices, all pointers are 32-bits wide as there is no advantage in allocating smaller pointers. However, the OCG compiler detects when a pointer only has one target, and side-steps the pointer completely, making it a direct access.

Since an OCG compiler knows exactly which registers are available at any point in the program and also which registers will be needed for every interrupt function in the program, it can generate code that maximizes register coverage, and minimizes stack use, code size, and the number of cycles required.
a higher code:

“With millions of e-mails flying through the air, security cannot be an afterthought.”

Scott Torzke  
V.P., Global Security Research In Motion

The BlackBerry® Enterprise Solution is the leading wireless platform for connecting mobile users to the customers, colleagues and information that drive business. The BlackBerry® platform is designed with security at the core, incorporating a security model that includes advanced security features for protecting wireless data flow and solution components.

This would not be possible without the cleanest source code. Coverity helps Research In Motion (RIM) achieve software of the highest integrity with superior technology that automatically identifies and prevents code defects and vulnerabilities.

Code integrity, it's the code we all live by.

www.coverity.com/3
DYNAMIC CONTEXT GENERATION

The compiler’s contribution to interrupt latency is the way it generates the code that switches contexts. The amount of context switching code is directly related to the number of registers it saves in response to an interrupt. Conventional compilers save every register that might be used by an interrupt because they have no way of knowing which registers will or will not be used by it.

For example, during high-priority interrupts that don’t take advantage of the shadow registers in the PIC32, conventional compilers always save all 32 registers, requiring many, for example, to require a total of 65 instruction cycles, compared with the OCG requirement of 22. This may not seem like much of a difference, but in an interrupt-intensive application, the CPU could spend thousands of extra cycles unnecessarily saving empty registers. Interrupts that don’t use the shadow registers typically require 20 instruction cycles for the context save/restore.

In contrast, because an OCG compiler knows exactly which functions call, and are called by, other functions, which variables and registers are required, and which pointers are pointing to which memory banks, it also knows exactly which registers will be used for every interrupt in the program. It can generate code accordingly, minimizing both the code size and the cycles required to save and restore the context. An OCG compiler, may save as few as three registers for a high-priority interrupts that don’t use the shadow registers, reducing the total number of required instruction cycles from 65 to just 22—reducing interrupt latency by nearly 70%! For interrupts that don’t use the shadow registers, the OCG compiler reduces interrupt latency by 10%.

Depending on the application, the cycle savings can be substantial (shown in Figure 5). When compiled by a conventional, non-OCG compiler, a simple benchmark program with 65,535 interrupts requires over 8,650,624 cycles for the PIC32 to execute at 80 MHz with two wait states.

The same program, compiled by an OCG compiler takes only 6,356,898 cycles—or 26.5% less (as Table 1 shows). In an interrupt-intensive program, the OCG compiler boosts the CPU’s performance by nearly 25%. A more interrupt-intensive program could see an even larger performance improvement.

STATIC VERSUS DYNAMIC CALLING CONVENTIONS

The large number of registers on the PIC32 provides a substantial opportunity for boosting CPU performance because function parameters and other data, usually stored in SRAM, can be stored in the registers that require fewer cycles to access. Efficiently exploiting the PIC32’s 32 registers can reduce the number of load/store cycles, freeing them up for computation and poten-
tially improving processor throughput.

How much the compiler “knows” about the register usage of a called function plays a big role in how efficiently the compiler exploits the PIC32’s registers. Conventional compilation technology does not have enough intelligence about the whole program to truly optimize register coverage because of the modular nature of embedded software and compilation. They must rely on rigid, static calling conventions that dictate how arguments are passed and values returned. Each calling convention contains a set of rules that defines which

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCG compiler</td>
<td>6,356,898</td>
</tr>
<tr>
<td>Non-OCG compiler</td>
<td>8,650,624</td>
</tr>
<tr>
<td>OCG difference</td>
<td>-27%</td>
</tr>
</tbody>
</table>

Table 1

Figure 5
CPU registers are to be preserved across calls. All functions in the program must adhere to the same calling convention.

The calling convention in GCC-based compilers specifies a fixed set of four PIC32 registers for passing parameters to functions. If a function requires fewer than four parameters, the compiler still considers all four registers to be used by parameters. They cannot be used for anything else. If the function requires more than four parameters, the extra parameters are passed on the stack in SRAM, even if other nonreserved registers are available. For example, if a function, function_1, calls another function, function_2, a conventional compiler will put function_1’s parameters on the stack to make room for function_2’s parameters, which will be loaded to the four registers specified by the calling convention. Other registers may be available, but they will not be used unless they have been specified in the calling convention for this purpose. When the second function is complete, the parameters from the first function also may be retrieved from RAM and loaded back into the registers. This cycle- and code-wasting data shifting will occur because the calling convention says it must be so.

In a sample program, with function calls nested three deep and each function having six parameters, a non-OCG PIC32 compiler uses four registers for passing parameters between functions as required by the calling convention. It frequently moves parameters between the four registers and the stack. This data shifting consumes 144 bytes of stack space in SRAM and generates 476 instructions that require 118 CPU cycles to move the data between the registers and the stack every time it happens.

Rather than relying on static calling conventions to allocate data between an unknown number of available registers, an OCG compiler waits until a view of the whole program is available.

**OCG dynamic register allocation.**

**Conventional compiler**

Register convention used for passing parameters

1) Load of parameters for function_1

2) Allocation of autos for function_1

3) Registers to call function_2 made available by unloading registers onto the stack

4) Load of parameters for function_2

**PIC32 MCU family with OCG**

Most registers used for passing parameters

1) Load of parameters for function_1

2) Allocation of autos for function_1

3) Load of parameters for function_2

**Figure 6**

The PIC32 has such a large register set, that it is usually possible to com-
pletely avoid using the stack for function parameters. In the code example cited above, the OCG compiler determined that 18 registers were available for the nested functions at this point in the program. Accordingly, it allocated all the parameters to the registers and used the stack only for the return address, cutting stack usage to only 16 bytes of SRAM. Because the parameters remained in the registers as long as they were needed, 30% fewer instructions were required (336) and the number of clock cycles required to execute fell from 118 cycles to only 80 instruction cycles—32% less (see Table 2).

According to Computer Architecture, A Quantitative Approach (John Hennessy and David Patterson, Ap Professional, 1990. Now published by Morgan Kaufman), 30% of a RISC CPU’s cycles are spent moving data. If the smaller contexts and more flexible register coverage provided by an OCG compiler can cut that amount by one third (to 20%), the available number of cycles actual processing will increase by about 15% (from 70% to 80% of cycles). At 80MHz, this is equivalent to increasing the PIC32’s substantial 125 DMIPS capability to 145 DMIPS. In addition, since fewer instructions are generated to move data, the code size is smaller and the amount of SRAM required for the stack is also smaller, potentially allowing the use of a less expensive microcontroller with smaller flash and SRAM memories.

### MAINTAINING CODE PORTABILITY

Many engineers will be migrating code to the PIC32 from PIC24 or dsPIC devices. Over time, designers may develop really robust code on the PIC32 and migrate it down to less expensive MCUs to reduce costs in derivative end-products with different feature sets and price points.

Microchip has taken great care to ensure code portability between its 8-

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Registers</th>
<th>SRAM (kbytes)</th>
<th>Instructions</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCG compiler</td>
<td>18</td>
<td>16</td>
<td>336</td>
<td>80</td>
</tr>
<tr>
<td>Non-OCG compiler</td>
<td>4</td>
<td>144</td>
<td>476</td>
<td>118</td>
</tr>
<tr>
<td><strong>OCG difference</strong></td>
<td><strong>350%</strong></td>
<td><strong>-89%</strong></td>
<td><strong>-29%</strong></td>
<td><strong>-32%</strong></td>
</tr>
</tbody>
</table>

**Table 2**

**Connect with the leader in embedded USB software.**

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

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

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

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

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

**Micro Digital**

USB SOFTWARE LEADER

800.366.2491 sales@smxrtos.com

[www.smxrtos.com/usb](http://www.smxrtos.com/usb)
/16- and 32-bit PICs, with a common peripheral set and tool integration into its MPLAB IDE. The PIC32 instruction set architecture is extremely C-friendly and greatly reduces designers’ dependency on hand-crafted, non-portable assembly code. In addition, the PIC32’s linear address space and very generous memory resources make it unlikely that software engineers will be forced to resort to assembler to fit the design into scarce NVM or RAM—a common situation with 8-/16-bit MCUs.

However, if legacy PIC code has routines that use C extensions or have been written in assembly code, moving that code to the PIC32 can be extremely problematic. It might have to be rewritten from scratch. When migrating from a PIC32 to a PIC24 or dsPIC with separate memory regions, it may be difficult to avoid using C extensions or hand-crafted code to efficiently use the memories. Again, the compilation technology used to generate object code can play an important role here.

An omniscient code compilation solves this problem as well because it performs at compile time an analysis of the whole program, which it uses to make optimum decisions about memory placement, pointer scoping, and so forth, without any intervention from the programmer. It optimizes memory resources while eliminating any need for assembly code or C-language extensions. This means that legacy C-code code written for the PIC24 or dsPIC always remains straight ANSI C code and can be used as-is and recompiled for the PIC32 with minimal changes. Conversely code written for the PIC32 can be reused in 8-/16-bit PIC MCUs, using an OCG compiler.

Jeffrey O’Keefe is director of education and development at HI-TECH Software. He joined HI-TECH in 1997 as senior software engineer. In 1998 he received his PhD from La Trobe University, Australia, in digital signal processing and has undergraduate degrees in physics, mathematics, and electronics. You may reach him at jeff@htsoft.com

Matthew Luckman is chief technical officer of HI-TECH Software. He joined HI-TECH Software in 1998 and has a degree in information technology from Queensland University of Technology, Australia. You may reach him at lucky@htsoft.com.
The following article first appeared in the September 1989 issue of Embedded Systems Programming magazine.

Forth data structures

W
denever I hear some lout claim that Forth code is unreadable, I’m reminded of the tourist who, on returning from Paris, marveled at the French: “Their children are so smart! Why, even the youngest already speak French!” To the Forth programmer, mathematical expressions using reverse Polish notation are no odder than reading right to left is to an Arab; stack-based parameter passing is no stranger than case declension is to a speaker of Czech or Polish.

Having said all that, I must confess that Forth code isn’t always as readable as it could be. Often what makes Forth seem incomprehensible is not the language itself but a programmer’s poorly factored code or poorly conceptualized data structures. The latter is especially true of Forth programs for control applications. It doesn’t have to be this way, of course, and to prove it I’ll be presenting several examples of data-structure conceptualization.

All the examples discussed here follow the Forth 83-Standard, with particular reference to the “vanilla” Laxen and Perry F83 public-domain Forth system unless otherwise noted. I’ll presume the reader has a working knowledge of Forth and knows about the CONSTANT and VARIABLE words as well as the CREATE . . . DOES> construct. As we’ll be using examples from control applications, an occasional PC@ or PC! is thrown in to fetch and store bytes in the Intel 808x I/O space.

REAL-WORLD NIGHTMARES
A few years back I did some consulting for an automotive equipment engineer who had left the shelter of the corporate world for
the perils of private entrepreneurship. He had built a marvelously inexpensive and useful piece of automotive repair equipment, then mastered the trigonometric calculations necessary to achieve the desired results. After teaching himself assembly language and beginning to write the program, he switched to Forth to increase his personal productivity. He finally decided to hire a full-time Forth consultant to complete his masterpiece, which was “beginning to be difficult.” When we began factoring the code, I saw that the program had indeed become difficult; it started out with 10 screens of variable declarations!

2 CONSTANT CELL \ 16-bit processor 
| CELLS ( n1 --- n2) CELL * ;

and a word that can create other words:

: ARRAY ( #cell-entries ---) CREATE CELLS ALLOT DOES> ( index --- address) SWAP CELLS + ;

We then created the array and its access methods:

4 CONSTANT #FORCES
0 CONSTANT LEFT 1 CONSTANT RIGHT
2 CONSTANT UP 1 CONSTANT DOWN
2 CONSTANT LEFT 3 CONSTANT RIGHT
#SAMPLES #FORCES 2D-ARRAY FORCE

0 CONSTANT LEFT 1 CONSTANT RIGHT
2 CONSTANT UP 3 CONSTANT DOWN #FORCES ARRAY FORCE

Note that we used symbolic constants in place of numeric literals, even at the level of the number of elements we’re going to declare in our array. This practice ensures that future modifications to the program entail simply changing a few constant declarations rather than hunting through the code for a specific occurrence of a number.

We could then address the forces in easy-to-read syntax—LEFT FORCE @ or DOWN FORCE !—to make the code simpler and its intention clearer. With the addition of these arrays, we can get and store data using such phrases as START DOWN FORCE @ and MIDDLE RIGHT FORCE !. Note that declaring arrays in this fashion takes one-based arguments, but the indices to the arrays are zero-based.

TRIMMING THE OVERHEAD
Astute Forth programmers will realize that the indexing process just described consumes run-time cycles. In cases where the desired indices are known at compile time, this runtime overhead may be abolished without seriously impacting the readability of the code. If at some point in the code we’ll store the top stack item to the END UP FORCE, we write the phrase:

... [ END UP FORCE ] LITERAL ! ...

inside a colon definition to cause the array-address calculation to take place at compile time and be stored in the routine as a numeric literal.

HANDLING I/O DATA
I/O handlers can benefit from the use of symbolic constants as much as arrays. The statement 13 ECCO PC! for example, is hardly as readable as CNT-INIT COUNTER-0 PC!. Symbolic constants are often only the beginning, however. Depending on hardware constraints, your input and output routines may require special data structures.

When mapping an eight-bit port to several bit entities, you may wish to create routines that manipulate given bits without disturbing their neighbors. Many modern microprocessors have memory-mapped I/O along with bit-set and bit-clear operations in their instruction sets. Forth systems frequently have bit-set and bit-clear operations that, on such microprocessors, will be code words taking
Boldly go where no system has gone before....

EPIA FX-Series Pico-ITX Mainboard

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

16 Isolated Digital Inputs
16 Relay Outputs

16-Bit Analog Input 16-Channels 500kHz

15 Independent 16-Bit Counter / Timers

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
Listing 2  N-dimensional array.

```plaintext
\ vizual//\ dimarray.src ... defining word
\ Creates a self-indexing n-dimensional array with variable dimension size from stack parameters.
\ USAGE:
\  2 3 5 3 dimarray 3D-THING
\ (3D-THING is a three-dimensional array. Dimension 0 contains elements 0 through 1, dimension 1)
\ (contains elements 0 through 2, and dimension 2 contains elements 0 through 4.)
\  0 0 0 3D-THING
\ (will yield the address of the 0th element of the 0th dimension)
\  1 2 4 3D-THING
\ (will yield the address of the last element of the last dimension)
\ The only confusing point is that DIMARRAY is created with stack parameters that are one-based.
\ whereas the run-time is zero-based.

: dimarray \ size1 .. sizen #dims ---
create \ Make header.
dup \ Dupe the number of dimensions.
1+ 0 \ That number plus one is the number of indices to compile.
do
 i pick ,
loop
1
swap 0 \ For number-of-dimensions times...
do
 *

\ base multiplicand

\ multiply sizes.

\ That's array size above indices.
does> \ index1 .. indexn --- addr
swap \ Index n doesn't need a multiplier.
over \ object address to TOS
@ \ Get number of dimensions.
1- 0 \ That number minus one is the number of multipliers to obtain.
do
1 i 1+ 0 \ base multiplicand for n times
   do
   i 1+ cells 3 pick + @ *
   loop \ Prepare to shift multiplier out of the way...
   >r rot \ then bring run-time index to top.
   r> * \ Bring back multiplier.
   + \ Add product to cumulative index.
loop
\ That many cells...
swap
dup @ \ plus those consumed by the compiled indices...
1+ cells
+ \ yields the entry address.
+
```

: dimlims \ index1 .. indexn --TIB-- ---
, \ Get cfa of the coming DIMARRAY object.
dup >r \ Save copy of cfa.
cell + \ Find pointer to pfa (F83 version).
dup \ Get pfa (= number-of-indices field).
@ dup \ Get number of indices.
depth 2- < not \ Stack too shallow?
abort" array dimension error - stack depth"
0 \ for number-of-indices iterations
do
 i 1+ pick \ Get one run-time index.
over i 1+ cells + @ \ Get corresponding compiled index.
over > \ Is zero-based run-time less?
swap -1 > \ Is it nonnegative?
and not \ But if tests fail...
```

Listing 2 continues on page 40.
EE Times Edge. Life is better on the edge.

Seize this opportunity to get insight and intelligence only available with EE Times Edge.

Your free subscription awaits you for EE Times Edge, the enhanced digital edition of EE Times. Get exclusive access to the industry’s hottest stories, market intelligence, competitive information, and end-business news. Be the first to receive the global insight and inspiration you can only get on the edge.

**Upcoming, exclusive content includes:**

- Top-ten technologies behind the Beijing Summer Olympic Games
- Investment and growth in green electronics
- MEMS in the real world
- Weather forecasting technologies
- China’s home grown mobile phone system

And with your free subscription to EE Times Edge, you’ll get exclusive access to special offers, product teardowns, surveys, and much more you just can’t get anywhere else.

Visit [www.eetimes.com](http://www.eetimes.com) and subscribe to EE Times Edge for the advantage you need.
Listing 2, continued from page 38.

```
abort" array dimension error - index size"
loop
drop \ drop the pfa, but the cfa...
r> execute \ executes as usual.
;
\ USAGE: (with DIMARRAY from dimarray.src)
\ \ \ 4 4 5 6 4 DIMARRAY MY-4D-ARRAY
\ \ \ creates a four-dimensional array consisting of 4*4*5*6 cell-length entries organized into
\ \ \ a zero-based indexing system that yields possible indices between 0 and 3 for the first
\ \ \ dimension, 0 and 3 for the second dimension, 0 and 4 for the third dimension, and 0 and 5
\ \ \ for the last dimension in MY-4D-ARRAY. Now
\ \ \ 3 3 4 5 MY-4D-ARRAY
\ \ \ yields the address of the last entry in MY-4D-ARRAY, but typing
\ \ \ 3 3 5 7 MY-4D-ARRAY
\ \ \ also yields an address, though an invalid one. So the usage
\ \ \ 3 3 5 7 DIMLIMS MY-4D-ARRAY
\ \ \ will abort with an error message instead of yielding a spurious address. DIMLIMS will also
\ \ \ abort if any index is negative or on stack underflow.
```

Listing 3   I/O handling example.

```
E000 CONSTANT PORT-BASE \ base address
0 CONSTANT PORT-A \ offset ... port A is at E000
1 CONSTANT PORT-B \ offset ... port B is at E001
\ VARIABLE PORT-BYTES \ Keep port state. If more ports, use array.
80 CONSTANT ZIGGER \ Bit mask.
40 CONSTANT ZAGGER
-1 CONSTANT ACTIVE \ "comfy" symbolic names for flags suitable for OR/AND masking
0 CONSTANT INACTIVE

: SET-STATE ( hi|lo mask port ---)
   DUP >R \ Save a copy of the port offset.
   PORT-BYTES + C@ \ Fetch the current mask for that port.
   ROT \ Bring flag to top of stack.
   IF OR \ Hi desired? OR tn the bit mask.
   ELSE SWAP NOT AND \ Otherwise, AND in the one's complement.
   THEN
   DUP R@ PORT-BYTES + C! \ Store new mask to our variable.
   R> PORT-BASE + C! ; \ Output the data to port itself.
```
advantage of the bit instructions.

There is a catch, however, when you read the fine print: those instructions tend to be readwrite-modify cycles and therefore unacceptable for use with peripherals for which a read cycle would constitute a special event or a command (consider, for example, DUART chips such as Motorola’s MC68681 or the Signetics 2681).

Let’s say we have two ports, A and B, located at 0E000 and 0E001 in a memory-mapped I/O configuration. Each has a zigger on bit D7 and a zagger on bit D6. Listing 3 shows the resulting code. With the addition of the SET-STATE word, we can turn on the port A zigger without disturbing the adjacent zagger. Interpretively you’d use a phrase like ACTIVE ZIGGER PORT-A SET-STATE, which in a colon definition might look like this:

```
: CYCLE-ZAGGER-B ( ---) BLETCH-TEST
  IF ACTIVE ELSE INACTIVE THEN
  ZAGGER PORT-B SET-STATE ;
```

Of course, if BLETCH-TEST were a polite word that returned a pure, true flag (FFFF), it would be easy to AND the result directly with the bit values and omit the IF... THEN clause from that definition.

HANDLING TABLES

The final common data structure we’ll look at is the lookup table. Lookup tables are handy for avoiding time-consuming run-time calculations, such as trigonometric functions, or in cases where the data is provided in table form from a component manufacturer, such as an ohms/temperature table documenting a thermistor. Lookup tables may be used singly, if the interval between indices to lookup values is regular for one side of the equation, or in pairs if no convenient integer relationship can be demonstrated among the indexing values. Creating a table is relatively easy, requiring only a name, CREATE, and a list of data. Accessing the data elegantly is almost as easy.

Listing 4 is an example of a single lookup table. In this particular implementation, we left the original reading on the stack and placed on top of it the index of the first lookup that surpassed the reading. Those two items (in conjunction with future readings) could be used to interpolate intermediate readings. For example, if the data consisted of thermistor readings, the indices represented degrees centigrade from some basepoint, and we didn’t need interpolation, our conversion would be simple:

```
25 CONSTANT BASEPOINT
: TEMP ( reading -- temp)
  LOOKUP BASEPOINT + NIP ;
```

For some users—lab technicians, for example—interactive tests would be as easy as using phrases like CORE PROBE TEMP.

BOTTOM-LINE ANALYSIS

The moral of these examples is that Forth supports the creation of entirely user-specified data structures with minimal coding effort. A little time spent factoring your data and code words can work wonders in improving the readability and efficiency of your programs. It can also make your development appreciably faster. And best of all, it often translates into happier customers.

When this was written, Jack Woehr was a programmer at Vesta Technology Inc. in Wheat Ridge, Colo. He was also the International Chapter Coordinator for the Forth Interest Group and a GEnie Forth RoundTable sysop.
C/C++ Development Kit including best-in-class compilers, genuine Keil μVision®, and royalty-free RTX RTOS

ULINK2 Adapter for target debugging and Flash programming

Keil RTOS and Middleware components are specifically optimized for embedded systems and include TCP/IP, Flash File system, USB and CAN support

Call 1-800-348-8051 for a free demo CD

Redefining the User Experience

PEG® GUI Development Tools

A family of portable graphics software for designing a high performance GUI for any embedded device.

- Completely customizable
- Multi-lingual support
- High color depth support
- Small footprint
- Fast execution speed
- Designed for cross platform application development

Statement of Ownership, Management, and Circulation

Publication title: Embedded Systems Design
Publication number: 1558-2493
Filing date: 9/3/2008
Issue frequency: Monthly
No. of issues published annually: 12
Annual subscription price: $0.00

Complete mailing address of Known Office of Publication:
600 Community Drive, Manhasset, Nassau County, NY 11030-3875

Full Names and Complete Mailing Addresses of Publisher, Editor, and Managing Editor:
Publisher: David Blaza, 600 Harrison Street, San Francisco, CA 94107; Editor: Richard Nass, 600 Harrison Street, San Francisco, CA 94107; Managing Editor: Susan Rambo, 600 Harrison Street, San Francisco, CA 94107

Owner: UBM Media LLC (600 Community Drive, Manhasset, NY 11030-3875), an indirect, wholly owned subsidiary of United Business Media (Ludgate House, 245 Blackfriars Road, London SE1 9UY U.K.)

Known Bondholders, Mortgages, and Other Security Holders Owning or Holding 1 Percent or More of Total Amount of Bonds, Mortgages, or Other Securities: None

Issue Date for Circulation Data Below: October 2008

Extent and Nature of Circulation:

<table>
<thead>
<tr>
<th>Average No. of Copies Each Issue During Preceding 12 Months</th>
<th>No. Copies of Single Issue Published Nearest to Filing Date</th>
</tr>
</thead>
</table>
a. Total number of copies (Net press run) | 38113 | 37571 |
b. Legitimate Paid and/or Requested Distribution (By Mail and Outside the Mail) | 38113 | 37571 |
   (1) Outside County Paid/Requested Mail Subscriptions Stated on PS Form 3541 | 35690 | 35490 |
   (2) In-County Paid/Requested Stated on PS Form 3541 | 0 | 0 |
   (3) Sales Through Dealers and Carriers, Street Vendors, Counter Sales, and Other Paid or Requested Distribution Outside USPS | 407 | 401 |
   (4) Requested Copies Distributed by Other Mail Classes Through USPS (e.g., First Class Mail) | 0 | 0 |
c. Total Paid and/or Requested Circulation | 36979 | 35891 |
d. Nonrequested Distribution (By Mail and Outside Mail) (1) Outside county As Stated on Form 3541 | 1309 | 1287 |
   (2) In-county As Stated on Form 3541 | 0 | 0 |
   (3) Distributed through USPS | 292 | 0 |
e. Total Nonrequested Distribution (Sum d1-3) | 1601 | 1287 |
f. Total Distribution (Sum e and f) | 37698 | 37178 |
g. Copies Not Distributed | 415 | 393 |
h. Total (Sum of f and g) | 38113 | 37571 |
i. Percent Paid and/or Requested Circulation | 95.75 | 96.54 |

This Statement of Ownership will be printed in the November 2008 issue of this publication.

I certify that all information furnished on this form is true and complete. I understand that anyone who furnishes false or misleading information on this form or who omits material or information requested on the form may be subject to criminal sanctions (including fines and imprisonment) and/or civil sanctions (including civil penalties).

Signature and Title of Editor, Publisher, Business Manager, or Owner: David Blaza, Publisher, October 1, 2008.
On the outside, it looks a lot like an iPod and acts a lot like an iPod, performing the same functions as an iPod. But peel back the cover and Microsoft’s Zune reveals it’s not an iPod; Microsoft has gone to great lengths to configure the Zune in their own image.

The first-generation Zune, designed in partnership with Toshiba, on the inside at least, looked a lot like Toshiba’s own player. The second-generation player was defined from scratch by Microsoft, including all the hardware, software, and mechanics.

Microsoft did a good job specifying exactly what they were looking for and not settling for anything less. The Zune is designed with an easy-to-use touch sensor, a 1.8-inch color display, and a WiFi connection. And all those features are packed into a highly integrated device adds an FM radio and WiFi connectivity.
Tear Down

The Zune has a Freescale i.MX32 microprocessor; ARM1; a Marvell 88W8686 WiFi transceiver; an ASIC developed by Synaptics to handle the sensing; a Silicon Motion SM267 flash-memory controller; and Hynix Mobile DDR SDRAM and MLC NAND flash memory.

about 3.5 by 1.5 inches, a feat Microsoft achieved through an innovative board design that also left room for the battery.

The model that I took apart was powered by a Freescale i.MX32 microprocessor, as shown in Figure 1. While this series of microprocessors is in the Freescale general catalog, this particular version is not available on the open market. It’s only available to special high-volume customers.

Similar to an i.MX31 (now the standard product), the i.MX32 tries to balance high performance with low power. It’s designed with an ARM11 CPU core.

“For a product like the Zune that employs a feature-rich operating system like Windows CE, it’s important to have a lot of headroom in terms of performance on your host processor,” says Boris Bobrov, Freescale’s manager of applications engineering. “That way you don’t spend extra battery life for high performance.”

LOTS OF PROCESSING POWER

My immediate reaction to having an ARM11 in an MP3 player was that it’s overkill, and you don’t want to pay for processing power that you’re not using. But after further analysis, it became clear that the user interface on the Zune requires all the available performance, with its snappy, feature-rich graphics. That’s one of the features of the Zune that separates it from the competition.

“It’s pretty common that the requirements of the user interface drive the choice of the processor,” says Dan Loop, i.MX product manager at Freescale. “Being able to browse large content libraries is something that OEMs are pushing for in this class of device.”

The Zune’s different subsystems operate at a variety of voltage levels. One of the nice features of the i.MX processor is that it always defaults to the lowest voltage possible for the required performance level. To ensure maximum performance, the system designers take advantage of techniques like dynamic temperature compensation, which measures the processor’s temperature. Dynamic frequency scaling, depending on the CPU load, reduces the frequency, thereby reducing the voltage. The processor’s main frequency operating points are 66, 132, 266, and 532 MHz.

SOFTWARE CHALLENGES

The software side of the design presents its own challenges. For example, the design team had to integrate the multimedia codecs into Microsoft’s software environment. “There’s so much content out there that our customers have to support, that running the performance tests and dealing with tens of thousands of audio and video streams from various sources is a big challenge,” says Loop.

The codecs themselves are provided by Freescale and are optimized for the processor. Freescale then works directly with the OEM to make sure that the highest performance level is achieved in the final software environment.

In any audio-based design, there are some important factors to be aware of, such as the annoying audio pops and clicks that occur when you first turn on the player, pause a song, or adjust the volume. Far too many systems ship without eliminating this problem. In the case of the Zune, that particular element was addressed very late in the design cycle. Hence, a lot of optimization and scrambling had to take place toward the end of that development.
Twenty years in the making, the Embedded Systems Conferences have surpassed all others to become the largest global brand to connect electronics engineers and vendor companies in three major markets – North America, China, Taiwan and now India. Widely acclaimed as the gold standard in embedded training and technical development, the Embedded Systems Conference India will provide high-quality content, delivered by the industry’s most experienced and respected experts.

Roadshow:
- Hyderabad: September 25-26, 2008
- Pune: September 29-30, 2008
- Noida: October 7-8, 2008

Contact us today to find out more about exhibiting and sponsorship opportunities.

For exhibiting opportunities please contact:

SEAN RAMAN
Event Sales Director, TechInsights
sraman@techinsights.com
Phone: 415-947-6622

For attendee information, please visit our event website: www.embedded.com/esc/india
The audio optimization occurred later than it should have because the real focus was on power management. It was important to optimize the power consumption around the player’s various “on” modes, especially when running video.

“We worked closely on the software with Microsoft, because we had to integrate software from various places, including ourselves, Microsoft, and Freescale,” says Paul Wilson, a product-line manager for audio products at Wolfson. “It all had to integrate seamlessly. There are three different development teams there, and they all have to be in sync with revisions of both hardware and software. For us, it was about delivering a set of functionality on the agreed-upon schedule. For example, Microsoft wanted different elements of the design operational at certain times.”

NO POPS, NO CLICKS
That first-generation Zune employed a Wolfson codec and a Freescale power-management part. A Maxim chip was used just to suppress audio pops and clicks. The second generation targeted better audio performance than the first generation and is at least as good as the main competitor.

The Wolfson part, the 8350, eliminates most of the unwanted audio, but it’s not completely pop free. When you’ve got a capacitor in the line, there must be some element of charge-discharge. A lot of optimization went into the power-up sequence, and that involved some engineering trial and error, as this is somewhat of a black art.

From the hardware designer’s perspective, it’s vital to follow the placement and layout rules. And he must select the proper capacitor values. For example, capacitor values can fluctuate if the device gets hot or the voltage isn’t exactly what’s expected.

Wolfson provides the core-layer drivers, like the initialization code for its 8350. They then ensure that it operates seamlessly with the Freescale processor. That keeps Microsoft from having to write code at that level. Microsoft’s focus can be where it belongs, on bringing the whole system together.

Wolfson claims to have a new technology on the drawing board that includes an integrated charge pump. Hence, they can eliminate the need for external dc capacitors in the headphone mode, reducing cost and saving board space.

Wilson adds, “Another thing people forget is that these capacitors can have a lot of yield problems in production and if you eliminate them, you eliminate potential yield losses.”

FM RADIO
One of Zune’s features that I liked was the FM tuner. In this case, it was a Silicon Labs Si4703. The part makes use of a standard antenna. Alternatively, Microsoft could have opted for a tuner that used the headphone as the antenna, but that requires the platform to employ a wired antenna. Although a wired antenna is the typical user model for the Zune, you want to be designing with an eye toward the future and the potential for wireless headphones.

“This is something that we talked to Microsoft about extensively,” says Wade Gillham, a senior marketing manager for broadcast products at Silicon Labs. “We have a technology on our chip that lets you put an antenna inside the enclosure of a very small, dense device, like a Zune, and still get good FM performance.”

In general, dealing with radios that operate on different frequencies doesn’t present a problem for the FM radio. You may get the occasional spikes, but it’s more a function of the internal frequencies of the system’s ICs, and whether those ICs have emissions that could potentially interfere with the WiFi or Bluetooth signals at 2.4 GHz, among other issues.

With respect to the FM radio, the Microsoft designers had to decide whether to include a digital output in addition to the analog out. Digital output is becoming more common for recording FM and then using that information for time-shifting the content. Features down the road could include FM transmit (in addition to receive) and AM receive.

Gillham says, “Transmit allows you to have more of a sharing and social networking experience. You can use the same embedded antenna to do receive and then you change the chip’s mode to do the transmit. And AM has its own challenges. There are more challenges on the AM side because that band sits right where all the spurs tend to sit.”

The transmitters that are available today are typically analog solutions, and they’re prone to have different peaks across the FM band. To avoid violating FCC regulations, you must program your output power so that the peaks are either at the high or low end. But that results in a less efficient output in the other parts of the FM band.

The Si4703 is also designed with a digital compressor that gives the audio a richer sound. The part also uses less external components than competitive devices. For example, the VCOs and inductors are all integrated.

Rounding out the Zune is a Marvell 88W8686 WiFi transceiver; an ASIC developed by Synaptics to handle the sensors; a Silicon Motion SM267 flash-memory controller; and Hynix Mobile DDR SDRAM and MLC NAND flash memory.
We Build Tiny Web Servers

Low-Cost Web Servers

Web-Enable Almost Anything for Almost Nothing

By the Staff of Geist Technology

Request your free book at FreeBook@GeistTek.com or (512) 331-8676

FREE BOOK!

Low-Cost MCU

8051 C Compiler Family

FREEWARE

New PRO compiler also operates in
Lite mode with no time or size limits:
silabs.htsoft.com/portal/ESD-2D

New Hi-Tech C® PRO for the Silicon Labs

1.7 x 1.9

$189

Tiny, yet Linux capable

84MHz CalFire MCU
16MB fast SDRAM
4.5 MB flash memory
SD card socket (to 20B)
1 CAN port
LCD/KPD port
Eclipse CDT based IDE
Free serial debugger
55 I/O pins & capabilities to burn...

INTEC AUTOMATION INC.
www.steroidmicros.com

Free Book!

Read about the largest case study on peer code review ever published.
Free copy. Free shipping.
Visit our website today!

www.codecollab.com

SmartBear Software

Display with Touch Screen

Embedded, Standalone, C/C++ Programmable...
Low Cost User Interface with Ethernet, USB, CF, RS232, ADC, DAC, I/Os, Relays...

U-Drive™
5.7” QVGA TFT, CAN, Ethernet, CF, ADC, DAC, ....

60+ Low Cost Controllers with TFT, ADC, DAC, UARTs, 300 I/Os, solenoid, relays, CompactFlash, LCD, Ethernet, USB, motion control. Custom board design. Save time and money.

TERN INC.
1950 5th Street
Davis, CA 95616 USA
Tel 510-754-0180 Fax 510-754-0181
www.tern.com
sales@tern.com

Low Cost Panel PC

The PPC-E7 is a Compact Panel PC based on a 200 MHz processor ARM9 with the following features:

• Open Frame Design
• 10/100 Base Ethernet
• Real Time Clock
• I2S Audio Interface
• 3 USB 2.0 Host Ports
• Fanless ARM9 200MHz CPU
• 3 Serial Ports RS232/485 & SPI
• SD/MMC Flash Card Interface
• Battery Backed Real Time Clock
• Up to 64 MB Flash & 128 MB RAM
• GPIO, A/D, Timers & PWM
• Linux with Eclipse IDE or WinCE 6.0
• WVGA (160 x 480) 7” LCD with Touch

Since 1995
23 YRS IN BUS.
200 SERIES

EMAC, inc.
EQUIPMENT MONITOR AND CONTROL
1-800-999-1993
www.emacinc.com
Phone: (619) 529-4525 Fax: (619) 457-9110 Web: www.emacinc.com

PC Based Logic Analyzers
NEW Model DV3400
• 18 / 36 Channels
• Up to 400 MHz
• PC, SPI, RS-232, included.

DigiView

Professional Hardware Capture
Software Analysis

TechTools
www.techtools.com (773) 272-9326 sales@techtools.com

47
**Cogent CSB7xx - Low Power System On Module (SOM)**

Pin-Compatible 2.63” x 2” SODIMM Form Factor Featuring:
- 200Mhz to 600Mhz ARM9/11, XScale, PowerPC and Coldfire Cores
- 64-128Mbyte SDRAM, 64Mbyte NOR FLASH and 512Mbyte NAND FLASH
- VGA to XGA LCD Controllers (some with 2D/3D and MPEG Acceleration)
- 10/100 Ethernet, USB Host and Device (some 480Mbit)
- Multiple UART/SPI, SBI and I2C Busses for I/O Expansion
- Compact FlashHIDE and 4-Bit SDIO Controllers
- On-Board Wide Input (6V to 35V) Power Supply  
  + 770mW typ., 1.2W max., <50mW standby
- I/O Expansion via CSB908xx Modules
- Development Kits priced at $648 - $798, includes SOM and 480x272 4.3” Touch LCD

Cogent Computer Systems, Inc.  
**email: sales@cogcomp.com**  
**web: www.cogcomp.com**

---

**Ad index**

<table>
<thead>
<tr>
<th>Advertiser</th>
<th>URL</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCES I/O PRODUCTS INC</td>
<td><a href="http://www.accesio.com">www.accesio.com</a></td>
<td>37</td>
</tr>
<tr>
<td>ATMELE CORPORATION</td>
<td><a href="http://www.atmel.com/picopower/">www.atmel.com/picopower/</a></td>
<td>24</td>
</tr>
<tr>
<td>ALTERA CORPORATION</td>
<td><a href="http://www.altera.com">www.altera.com</a></td>
<td>CV4</td>
</tr>
<tr>
<td>CMX SYSTEMS INC</td>
<td><a href="http://www.cmx.com">www.cmx.com</a></td>
<td>34</td>
</tr>
<tr>
<td>COGENT COMPUTER SYSTEMS</td>
<td><a href="http://www.cogcomp.com">www.cogcomp.com</a></td>
<td>48</td>
</tr>
<tr>
<td>COVERITY INC</td>
<td><a href="http://www.coverity.com/3">www.coverity.com/3</a></td>
<td>29</td>
</tr>
<tr>
<td>EMAC INC</td>
<td><a href="http://www.emacinc.com">www.emacinc.com</a></td>
<td>47</td>
</tr>
<tr>
<td>EXPRESS LOGIC</td>
<td><a href="http://www.etos.com">www.etos.com</a></td>
<td>4</td>
</tr>
<tr>
<td>GREEN HILLS SOFTWARE INC</td>
<td><a href="http://www.ghs.com">www.ghs.com</a></td>
<td>1</td>
</tr>
<tr>
<td>HI-TECH SOFTWARE</td>
<td><a href="http://www.silabs.htsoft.com/portal/ESD-20">www.silabs.htsoft.com/portal/ESD-20</a></td>
<td>47</td>
</tr>
<tr>
<td>IBM IT</td>
<td><a href="http://www.ibm.com/green/software">www.ibm.com/green/software</a></td>
<td>6-7</td>
</tr>
<tr>
<td>INTEC AUTOMATION INC</td>
<td><a href="http://www.steroidmicros.com">www.steroidmicros.com</a></td>
<td>47</td>
</tr>
<tr>
<td>INTEL</td>
<td><a href="http://www.intel.com/go/rethink">www.intel.com/go/rethink</a></td>
<td>17</td>
</tr>
<tr>
<td>IT WATCHDOGS INC</td>
<td><a href="http://www.geistek.com">www.geistek.com</a></td>
<td>47</td>
</tr>
<tr>
<td>KEIL SOFTWARE</td>
<td><a href="http://www.keil.com">www.keil.com</a></td>
<td>42</td>
</tr>
<tr>
<td>MENTOR GRAPHICS</td>
<td><a href="http://www.mentor.com/nucleus">www.mentor.com/nucleus</a></td>
<td>CV3</td>
</tr>
<tr>
<td>MICRO DIGITAL</td>
<td><a href="http://www.smartos.usb">www.smartos.usb</a></td>
<td>33</td>
</tr>
<tr>
<td>MOUSER ELECTRONICS</td>
<td><a href="http://www.mouser.com">www.mouser.com</a></td>
<td>8</td>
</tr>
<tr>
<td>NATIONAL INSTRUMENTS</td>
<td><a href="http://www.ni.com/greenengineering">www.ni.com/greenengineering</a></td>
<td>11</td>
</tr>
<tr>
<td>PERFORCE SOFTWARE</td>
<td><a href="http://www.perforce.com">www.perforce.com</a></td>
<td>19</td>
</tr>
<tr>
<td>RADIAN HEATLINKS</td>
<td><a href="http://www.radianheatinks.com">www.radianheatinks.com</a></td>
<td>34</td>
</tr>
<tr>
<td>SMART BEAR SOFTWARE</td>
<td><a href="http://www.CodeCollaborator.com">www.CodeCollaborator.com</a></td>
<td>27</td>
</tr>
<tr>
<td>SMART BEAR SOFTWARE</td>
<td><a href="http://www.codecollab.com">www.codecollab.com</a></td>
<td>47</td>
</tr>
<tr>
<td>SWELL SOFTWARE</td>
<td><a href="http://www.swellsoftware.com">www.swellsoftware.com</a></td>
<td>42</td>
</tr>
<tr>
<td>SYMBIAN SOFTWARE LTD</td>
<td><a href="http://www.developer.symbian.com">www.developer.symbian.com</a></td>
<td>15</td>
</tr>
<tr>
<td>TECH TOOLS</td>
<td><a href="http://www.tech-tools.com">www.tech-tools.com</a></td>
<td>47</td>
</tr>
<tr>
<td>TECHNOLOGIC SYSTEMS</td>
<td><a href="http://www.embeddedARM.com">www.embeddedARM.com</a></td>
<td>51</td>
</tr>
<tr>
<td>TERN INC</td>
<td><a href="http://www.tern.com">www.tern.com</a></td>
<td>47</td>
</tr>
<tr>
<td>TEXAS INSTRUMENTS</td>
<td><a href="http://www.ti.com/5xx">www.ti.com/5xx</a></td>
<td>2</td>
</tr>
<tr>
<td>THE MATHWORKS</td>
<td><a href="http://www.mathworks.com/mbd">www.mathworks.com/mbd</a></td>
<td>CV2</td>
</tr>
</tbody>
</table>

**Embedded Systems Design**

**ADVERTISING SALES**

**MEDIA KIT: www.embedded.com/mediakit**

**Management**

TechInsights  
600 Harrison St., 5th Flr.  
San Francisco, CA 94107

James Lonsdale-Hands  
Vice President  
TechInsights Events  
(214) 415-0274  
jlonsdalehands@techinsights.com

**Emerging Accounts**

Steve Corrick  
Vice President of Sales  
(415) 947-6651  
sorrick@techinsights.com

**Advertising Coordination and Production**

United Business Media  
600 Community Drive  
Manhasset, NY 11030

**Pete C. Scibilia**  
Production Manager  
(516) 562-5134  
pscibili@ubm-us.com

**Advertising**  

Bob Dumas  
Associate Publisher  
TechInsights  
(516) 562-5742  
bdumas@techinsights.com

**TechInsights**  
600 Community Drive  
Manhasset, NY 11030

**National Sales Mgr.**  
Joanna Earl  
(785) 838-7560  
jearl@techinsights.com

**TechInsights**  
601 West 6th St., Ste B  
Lawrence, KS 66049

**United Business Media**  
600 Community Drive  
Manhasset, NY 11030

**Production Manager**  
Pete C. Scibilia  
Production Manager  
(516) 562-5134  
pscibili@ubm-us.com
The Agile Manifesto reads, in part, that the signatories value “individuals and interactions over processes and tools.” The field of management was revolutionized by “empowering” people to make decisions and take charge of their work, which led companies to realize the importance of hiring wisely.

People matter.

In my experience, most companies do prize their engineers. No, they’re not given CEO-like rock-star status, and we all wish salaries were higher. But engineers do make a decent middle-class wage, and in these days of nearly full employment in our industry, businesses fearful of losing their developers generally treat the engineers well. I do hear some dramatic exceptions, but most correspondents claim to be satisfied with their jobs.

But there are persistent complaints that never seem to go away. The first of these is overtime; the 40-hour work week is but a dream to many, and panicked overtime is de rigueur at many outfits in the last few months of a project.

Tired people make mistakes. You’d think it would be self-evident that overtime leads to buggier products. Or that safety issues in dangerous environments will escalate when workers take a shortcut that they’d never consider in a less sleep-deprived state.

In my collection of embedded disasters, a common theme is tired engineers. Investigating bodies routinely cite “60 to 80 hours of work a week for months on end” as a contributing cause to a system failure. And, since only the most expensive systems, like space missions, are investigated when something goes wrong, the dollars lost when a weary worker enters the decimal point in the wrong place are staggering.

Most professions suffer from the curse of OT. Doctors get midnight calls from the hospital, lawyers work late in the night to respond to an unexpected judicial ruling, and accountants dread the arrival of tax season. Is this good? I doubt it, and mounting evidence suggests medical mistakes, no doubt at least partially driven by tired people, kill.

Maybe we should plan better, but even the best planning fails when something unexpected occurs. And the unexpected is one of the things we should expect most when designing a new product.

“Unexpected” sometimes comes because we’re confused about R&D. One of my top ten reasons why projects fail is because the science is bad: engineers are developing the science in parallel with making the product. For instance, the algorithm changes constantly as we play with the system’s chemistry to get meaningful data. That’s a sure-fire route to scheduling disaster, and often results in cancellation of the project. I have seen companies go out of business because engineering is so wrapped up in R&D they never get a product to market.

We don’t do R&D—or, if we do, we shouldn’t. It’s either R, or it’s D. Sure, development contains an element of research but is mostly about achieving a pretty well-understood objective using known science or algorithms.

It’s possible to plan D; no one can schedule R since that intrinsically explores the depths of the unknowns. If research could be scheduled, we’d know when the cure for cancer would be available.

Although it’s possible to plan development, it’s impossible to be perfect, so overtime will never go away. Wise managers, though, understand the costs.

---

Experts tell us interesting facts about our work lives that we may know but deny: programmers have human limitations.

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.
Turn R&D into ROI
before you hear from your CEO.

Now you can reach your audience faster, easier, and more affordably than ever before.

TechInsights™ Direct combines the media power and reach of *EE Times*, *TechOnline*, and *Embedded Systems Design* with the convenience of online buying and service. Take advantage of optimized media bundles specifically designed for your marketing objectives. Programs designed for new product promotion, lead generation, and new market penetration start as low as $7,500. And TechInsights Direct is designed to handle all your marketing initiatives including thought leadership, print and online ads, newsletters, webinars, technical papers, and more.

Let TechInsights Direct provide you with the marketing tools you need to succeed. After all, the last thing you need is the CEO knocking on your door.

To learn more, visit us at: [www.techinsights.com/direct](http://www.techinsights.com/direct)
Circadian's *Shiftwork Practices 2005* survey found that productivity can decrease by as much as 25% when workers put in 60+ hour weeks. Clearly overtime leads to diminishing returns for everyone involved.

The same survey showed that turnover is nearly three times higher among those working long weeks. Consider that a headhunter might charge a third of a year's wage to find a replacement, and before retraining costs, each person lost represents $30,000 or more out the door.

I've worked with more than a few self-described superprogrammers who couldn't code their way out of main().

Absenteeism is twice the national average at companies that routinely resort to long weeks. Stress leads to sickness, and even people shackled to the desk need free time to get all of the routine activities of life done.

Although overtime will always be part of the fabric of our profession, some toxic companies use it as a cost-saving strategy. Unless fairly compensated, that's servitude we should all reject. Engineers sell their skills to their employer, and their inventory is time. The company strategically translates their inventory into revenue. So should we.

**SUPERPROGRAMMERS**

What about superprogrammers? They are the folk that some companies rely on to defy all productivity statistics and crank a lot of code fast. I've been fortunate to work with some brilliant developers who hole up in their office and create wondrous products, fast, and without a fuss. They love to build stuff, eschew office politics, and take great pride in their work.

In a private study conducted for IBM in 1977, Capers Jones found that the best developers are about six times more productive than the worst. That's a pretty impressive number, but holds only for small (1,000 lines of code) projects. The difference decreases quickly as projects grow in size, till, at half a million lines of code, the best and worst developers are equally awful. Big systems require a lot of collaboration, so we spend much of our time on meetings, e-mail, reports, and more meetings. A superprogrammer and the worst person on the team attend meetings at exactly the same speed.

Tom DeMarco looked at superprogrammers. In *Controlling Software Projects*, he suggests that developers should specialize, an idea that somewhat mirrors IBM's old concept of structuring a team around a chief programmer. Although Joe might be an incredible designer, that's no reason to suspect he's any good at debugging. Perhaps Jill can

---

**7" Touch Panel Computer**

for embedded GUI / HMI applications

- Low power, Industrial Quality Design
- 200 MHz ARM9 CPU
- Power by a
- Programmable FPGA - SK LUT
- 512MB Flash w/ Debian Linux
- 7" Color TFT-LCD Touch-Screen
- 800x480 customizable video core
- Dedicated framebuffer - 8MB RAM
- Audio codec with speaker
- Runs Eclipse IDE out-of-the-box
- Boots Linux 2.6 in about 1 second
- Unbrickable, boots from SD or NAND
- Runs X Windows GUI applications
- Over 20 years in business
- Never discontinued a product
- Open Source Vision
- Our engineers can customize for your LCD
- Engineers on Tech Support
- Custom configurations and designs w/ excellent pricing and turn-around time
- Most products ship next day
- Over 20 years in business
- Never discontinued a product
- Open Source Vision
- Custom configurations and designs w/ excellent pricing and turn-around time
- Most products ship next day

Powered by a
200 MHz ARM9 CPU

- Low power, Industrial Quality Design
- Mountable aluminum frame
- 64MB SDRAM (128MB opt)
- Programmable FPGA - 5K LUT
- 7" Color TFT-LCD Touch-Screen
- 800x480 customizable video core
- Dedicated framebuffer - 8MB RAM
- Audio codec with speaker
- Boots Linux 2.6 in about 1 second
- Unbrickable, boots from SD or NAND
- Runs X Windows GUI applications
- Runs Eclipse IDE out-of-the-box

- Over 20 years in business
- Never discontinued a product
- Open Source Vision
- Custom configurations and designs w/ excellent pricing and turn-around time
- Most products ship next day

---

*www.embedded.com* | *embedded systems design* | *OCTOBER 2008* 51
clearly. Software development and writing convey information concisely and differenctly between DRAM and and certainly without a clue about the English majors! They’ll often graduate with no programming experience at all and take enormous personal satisfaction out of building much of the system.

CS, EE, OR BA?
If you can’t write it down in English, you can’t code it. So does that imply English majors might have a role in development?

When hiring firmware developers I’ve found little difference between CS, CE, and EEs for doing firmware development. In some industries, though, the degree might be important. One company I know hires mechanical engineers exclusively. They, too, get quite a bit of programming experience in college. Their sound understanding of machine control is what’s important to this company that makes material handling equipment.

ME, EE, CS, CE—all are engineering or engineering-like programs that stress practical problem-solving abilities. We’re taught to build things, to decompose big problems into solvable chunks. That in a large sense differentiates us from our friends studying the liberal arts.

Yet, in my career, I’ve found that some of the best developers of all are English majors! They’ll often graduate with no programming experience at all and certainly without a clue about the difference between DRAM and EPROM.

They can write. That’s the art of conveying information concisely and clearly. Software development and writing are both the art of knowing what you’re going to do and then lucidly expressing your ideas. The worst developers, regardless of background, fail due to their inability to be clear. Their thoughts and code tend to ramble rather than zero-in on a solution.

It’s easier to train someone in a new language than to teach them to think clearly. C really isn’t that hard to learn; it has but a handful of constructs. Most folks can learn the fundamentals in a week. Debugging takes longer, but all new programmers find themselves at sea when first faced with bugs. What do I do now? Should I single step through the entire program? How do I decide where to set breakpoints?

Too many engineering-trained developers have a total disregard for stylistic issues in programming. Anything goes. Firmware is the most expensive thing in the universe, so it makes sense to craft it carefully and in accordance with a standard style guide. That is: make sure it clearly communicates its intent. This is where the English majors shine; they’ve spent four years learning everything there is to know about styles and communication. And they’re used to working to a standard, like the Chicago Manual of Style.

HEROES
In a Dilbert cartoon, the pointy-haired boss, apparently frustrated by the company’s sub-par products, announces that he’ll reward each bug fix with a $10 bill. Wally says: “Hooray! I’m gonna code me a minivan!”

Unfortunately, the heroes, those who seem to save the organization in a great flurry of activity, are often reacting dramatically to the problems they created. Like Wally, they’re rewarded for the successes while no one notices that furious activity is no substitute for doing things carefully.

Solving problems is a high-visibility process; preventing them is much better, but earns few rewards. This is illustrated by an old parable:

In ancient China, there was a family of healers, one of whom was known throughout the land and employed as a physician to a great lord. The physician was asked which of his family was the most skillful healer. He replied, “I tend to the sick and dying with drastic and dramatic treatments, and on occasion someone is cured and my name gets out among the lords. “My elder brother cures sickness when it just begins to take root, and his skills are known among the local peasants and neighbors.

“My eldest brother is able to sense the spirit of sickness and eradicate it before it takes form. His name is unknown outside our home.”

Unfortunately, sometimes the very best developers get the least acknowledgement, even from their own teams. ■

ENDNOTES:
Nucleus® OS
At Your Fingertips

Define
Develop
Differentiate

the Nucleus OS advantage

For more information visit www.mentor.com/nucleus or call 1.800.547.3000

©2008 Mentor Graphics Corporation. All rights reserved. Mentor Graphics is a trademark of Mentor Graphics Corporation.
Low power

Highest functionality in its class

First 65-nm low-cost FPGA

VERY COOL

Cool off your system with Altera® Cyclone® III FPGAs. The market’s first 65-nm low-cost FPGA features up to 120K logic elements—2X more than the closest competitor—while consuming as little as 170 mW static power. That’s an unprecedented combination of low power, high functionality, and low cost—just what you need for your next power-sensitive, high-volume product. Very cool indeed.

www.altera.com/cyclone3

Copyright © 2008 Altera Corporation. All rights reserved.