An insider's view of the 2008 Embedded Market Study

The results are in. Here is an analysis of embedded systems industry's most comprehensive annual study.

BY RICHARD NASS

The results of the 2008 Embedded Market Study are now in. In some cases, the results are exactly what you might expect, and in others, they're quite startling. Recently Contributing Editor Michael Barr and I sifted through the data and discussed what it all means. This article shares our thoughts and analyses of why the results are what they are.

If you're not familiar with our annual study, here's the scoop. Earlier this year, we (*Embedded Systems Design* magazine) sent the survey out to a select list of our readers and people who attended one of the Embedded Systems Conferences. There's a good chance that you were one of the folks who received the study. About 1,100 people responded, which makes it a fairly representative sample of the embedded systems market.

If you filled out the survey, you know that it was quite comprehensive. Depending on how you responded to the questions, you may have had over 50 questions to answer. What I always find interesting about the Embedded Market Study is the year-on-year data, showing the trends from one year (or multiple years) to the next.

First I'll go through the profile of the respondents. The top ten application areas you're working on are (in order): industrial control, video and imaging, consumer electronics, aerospace, automotive, medical, military, computers and peripherals, data communications, and telecommunications.

Your job functions include writing software/firmware, debugging software/firmware, integrating hardware/software, selecting or specifying architecture, designing or analyzing firmware/software, managing projects, and debugging hardware.

More than half of you are working on a product upgrade, rather than a new project. More than half of those upgrades are taking advantage of a new microprocessor, hence requiring software changes.

For those projects that include a wireless ca-

Which of the following challenges are your own or your embedded design team's greatest concern regarding your current embedded systems development process?

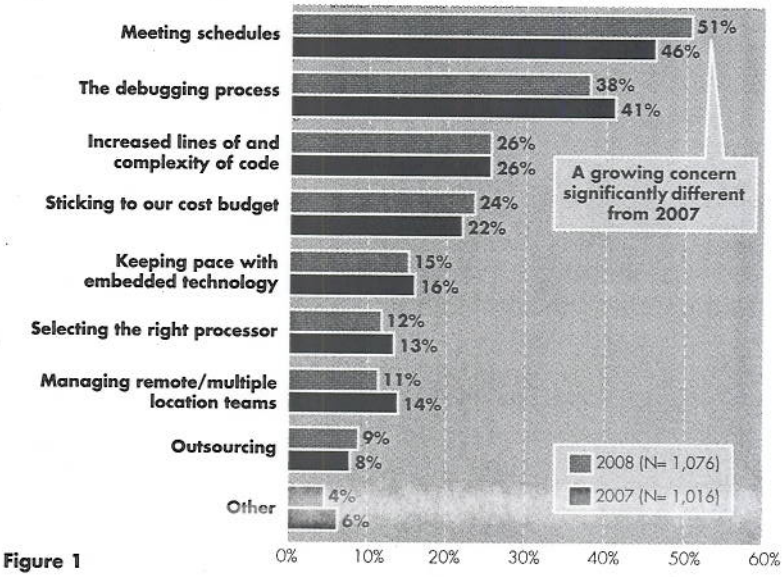

Figure 1

pability, more than half use Wi-Fi as the connection medium. That's up about 20% from 2007. ZigBee is also up about 20%.

The size of the average design team increased slightly, from 13.6 people to 15.2 people. But it's interesting to note that the number of software engineers on the team increased by almost two, meaning the number of hardware engineers stayed the same or was slightly reduced.

STILL WORRIED ABOUT DEADLINES

As shown in **Figure 1**, meeting schedules is still the number one concern for developers. In fact, that concern actually increased by about 10% over last year.

Figure 2 shows the environment that developers are operating in, for both their current and next projects. Not much has changed from last year's results to this year, which is a little surprising; I would expect last year's "next project" to be equal to (or at least near) this year's "current project," But that's obviously not the case.

Michael Barr noticed that "UML adoption remains extremely low at 16%, with no expected upturn. This is disappointing after so many years of pushing by so many people and companies."

That takes us to the question, "Which software/hardware tools are your favorite/most important? This is seen in **Figure 3**. Here, as Barr points out, "UML tools are the favorite/most important tool for only 6% of respondents. In addition, source-code analysis remains woefully underutilized, especially in light of the risks of bugs in deployed systems and the value these tools provide in that regard. Source-code analysis tools are 'most important' for only 7% of the developers."

The overwhelming favored tools are the compiler/assembler and the debugger, with the oscilloscope a distant third. It's also an interesting anomaly that software testing tools are way at the bottom of the list, yet users say that "test" in general is one of the key factors in the design process often taking the majority of the design time. That leaves me to conclude that users aren't happy with their current software test tools.

CODE REUSE IS HAPPENING, REALLY

Figure 4 puts a smile on my face. I've always preached the need to reuse code whenever possible, rather than writing from scratch. Too often, developers want to do it "their way." This self-indulgence may offer a slight improvement over the existing code (or it may not), but it almost always causes the project to take longer, even past the original deadline in some cases. The percentage of users who use new code all the time went from 15% down to 11%.

One of my favorite questions, shown in **Figure 5**, asked survey takers to rank which areas of the embedded systems industry had seen the most dramatic changes over the past 20 years and would continue to change the most over the next 20. Even with all the technology

My current embedded project uses . . .

My next embedded project is likely to use . . .

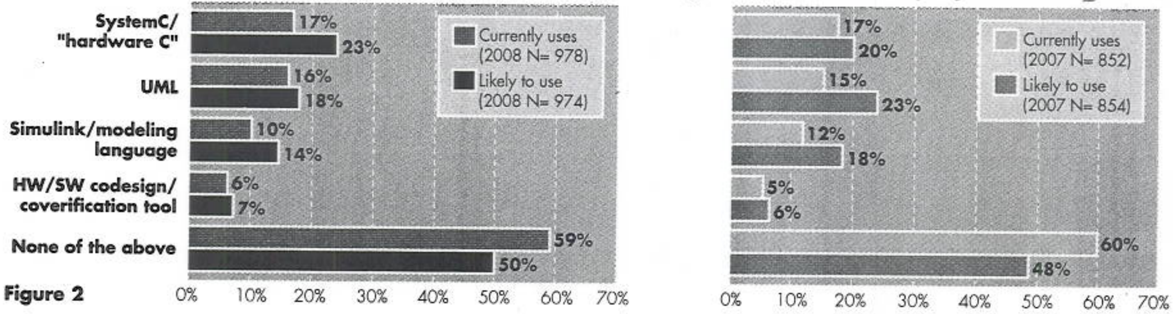

Figure 2

Which of the following are your favorite/most important software/hardware tools?

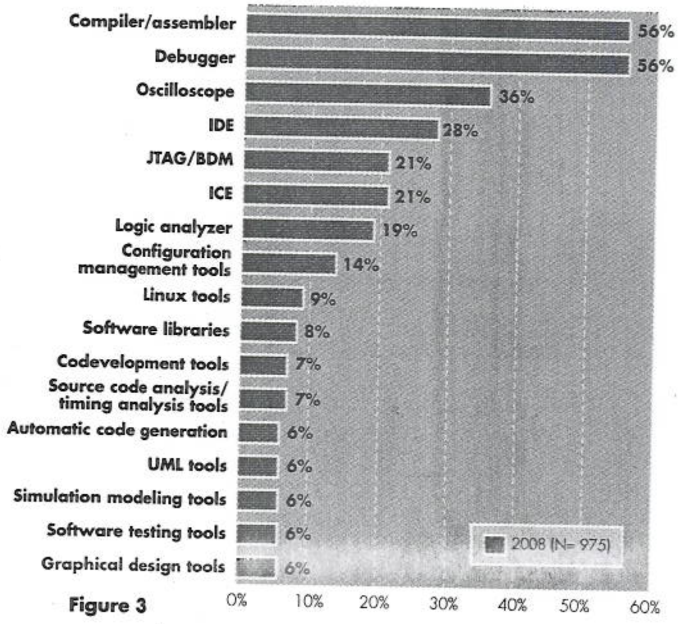

Figure 3

Does your current project reuse code from a previous embedded project?

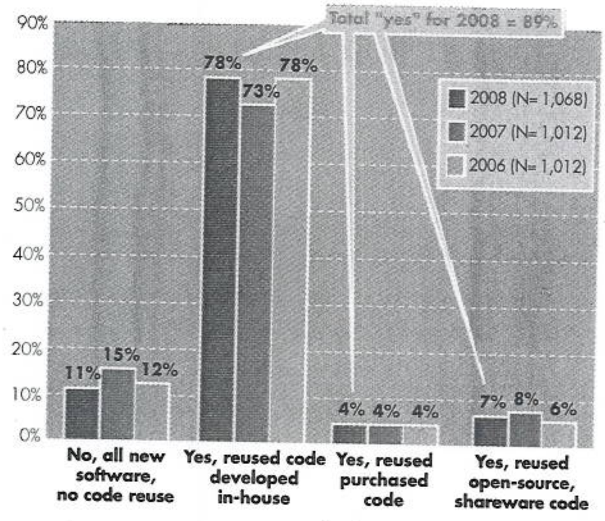

Figure 4

breakthroughs that we've seen in semiconductor design, more than half of developers still see big changes coming in chip technology. It's no surprise that we expect to see many more changes in global markets over the next 20 years than the previous 20.

Another noteworthy data point is the time-to-market. As if we didn't have enough problems (and short enough design windows), developers think that "speed to market" will dramatically change going forward. But how will time-to-market decrease if labor-saving methods, such as code reuse, are not adopted.

Barr looked at this question a little differently than I did (half-empty verses half full?). His take was that "it's depressing that 'professionalism and standards' (really the lack thereof) have not been considered to have

Reflecting over the past 20 years and looking to the next 20 years, select the areas below that you think have seen the most dramatic changes?

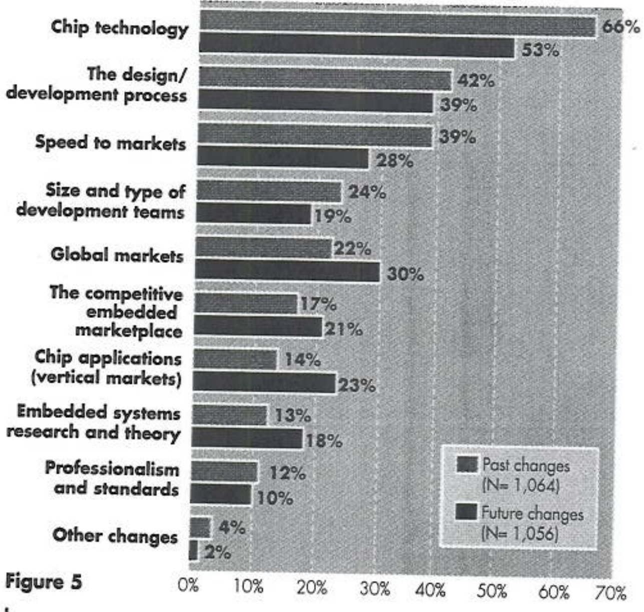

Figure 5

If current embedded project does not use an operating system, RTOS, kernel, software executive, or scheduler of any kind, why not?

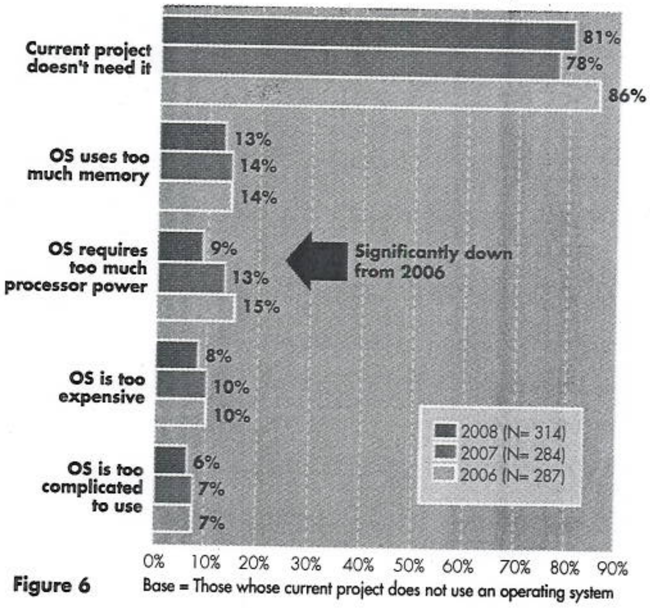

Figure 6

Base = Those whose current project does not use an operating system

My current embedded project uses . . .
My next embedded project will likely use . . .

Operating system currently use	Current project 2008 (%)	2007 (%)	2006 (%)	2005 (%)
Commercial OS	49%	47%	51%	55%
Open-source OS without commercial support	19%	22%	16%	25%
Internally developed or in-house OS	21%	21%	21%	20%
Commercial distribution of an open-source OS	11%	10%	12%	•

-6% in 4 years

Operating system plan to use next	Next project 2008 (%)	2007 (%)	2006 (%)	2005 (%)
Commercial OS	37%	41%	47%	50%
Open-source OS without commercial support	26%	27%	19%	34%
Internally developed or in-house OS	23%	15%	17%	16%
Commercial distribution of an open-source OS	15%	16%	17%	•

-13% in 4 years

• Indicates response option not asked in 2005 Base = Use/plan to use an operating system

2008 (N= 764) 2007 (N= 676) 2006 (N= 727) 2005 (N= 1,303)

Figure 7

Which factors most influenced your decision to use a commercial operating system?

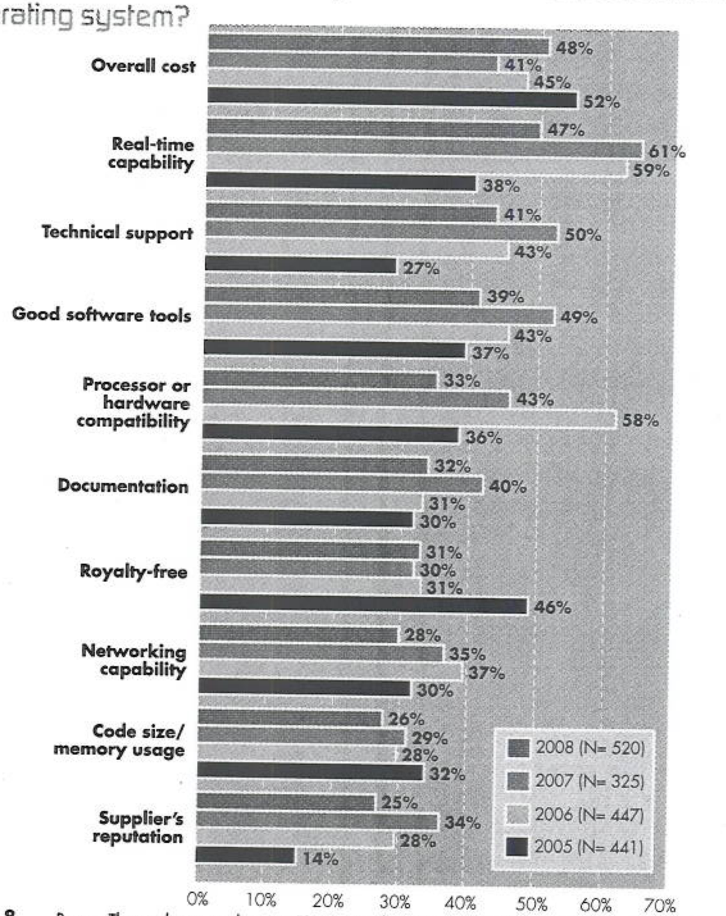

Figure 8 Base = Those who currently use a "Commercial" or "Commercial distribution" operating system

changed much over the past 20 years. And worse yet, the forward expectations are even lower for change here."

OPERATING SYSTEMS

As you might expect for a survey of embedded developers, there were lots of questions pertaining to operating systems. One that caught my eye was "If your current embedded project doesn't use an operating system (OS), real-time OS (RTOS), kernel, software executive, or scheduler of any kind, why not?" This is shown in **Figure 6**.

The top answer—my project didn't require it—didn't come as a surprise, and it pretty much remained the same from previous studies. But the response I found interesting was "the OS requires too much processing power." This number is down from last year, which was down from the year before. That's a good thing, as it shows that users are taking advantage of the performance offered by the processor vendors. Those vendors always talk about how much performance they offer, and it seems like users are jumping onto the bandwagon, even if it's in small increments.

Barr adds, "It's nice that 'too expensive,' 'too complicated,' and 'too much memory' are all going away as excuses."

The use of commercial operating systems, shown in **Figure 7**, reveals an interesting trend. Here, we show data from the previous four years, as it might be misleading to just look at this year versus last year. The four year trend of commercial OSes that developers plan to use is going down (even though it's up slightly in 2008 when compared with 2007). There's also a significant drop in the use of noncommercial OSes. That should certainly raise a red flag with the commercial OS vendors.

Barr was also quick to spot this trend, one he dubbed "very, very interesting. And Linux/open source doesn't explain that change—either in the unsupported or commercially sold variants."

For those who do employ a commercial operating system, we asked them why they did so, as shown in **Figure 8**. Frankly, this question left me scratching my head more than any other. From 2005 to 2007, the response "overall cost" declined. However, this year, it increased. That's likely a result of the poor economy we're mired in. A huge decline comes in "real-time capability." Where I'm really left wondering is why there would be significant declines in real-time capability, tech support, good software tools, and processor or hardware compatibility.

"The piece of data here that's potentially significant is the processor or hardware compatibility," explains Jack Ganssle, industry consultant and regular contributor to *Embedded System Design* and Embedded.com, whom I consulted to get a better understanding of these responses. Jack said, "I can read this data in one of a few ways, but one take on this is the decreasing importance of instruction set architectures. X86, PPC, ARM—who really cares? It's all in C so things are pretty compatible."

Next, we asked what the most important factors in selecting any OS are (as opposed to just commercial OSes), as you can see in **Figure 9**. Again, I scratch my head, because the answers are inconsistent with the previous question (**Figure 8**). It does, however, seem to validate the data. Barr claims that the excuses for not using an RTOS are going away.

When it comes to choosing a microprocessor, **Figure 10** shows that the hardware team has the most influence, but the software staff get their two cents in as well. But, oddly enough, teams that make a group decision are in decline by about 10%. Comparing these results to those of another question (not shown here, "Who has the most influence on the choice of operating system"), it appears that software developers have more say in the hardware than hardware developers do in the software. Go figure.

OUTSOURCING

Then there's the dreaded "O" word—outsourcing. In lots of circles, this is a dirty word, because it's often closely

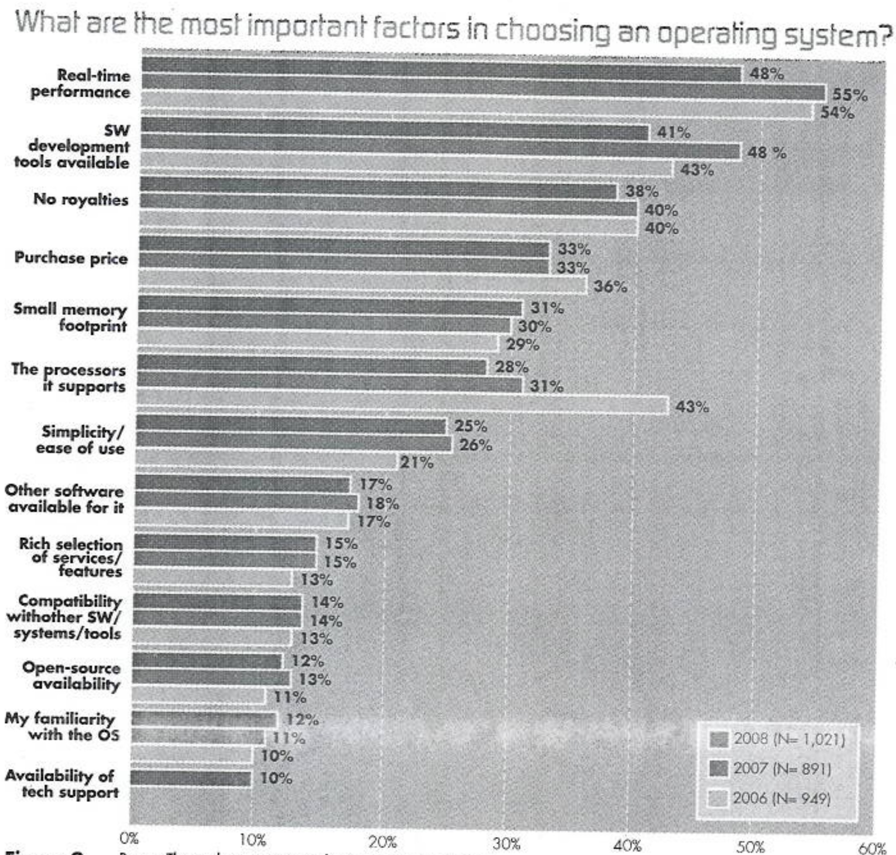
What are the most important factors in choosing an operating system?

Figure 9 Base = Those who are using or plan to use operating system

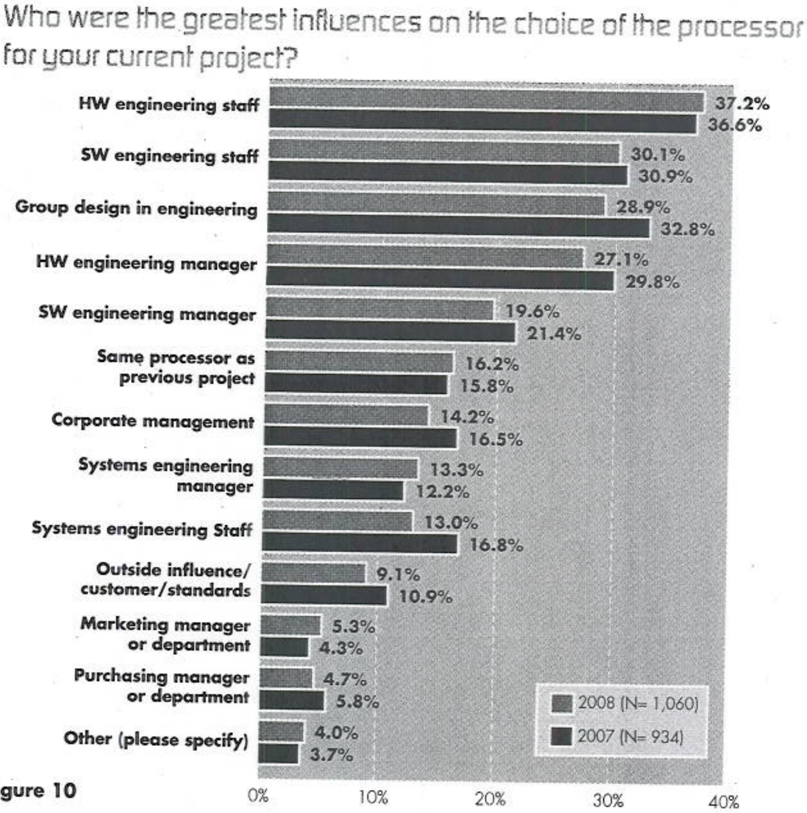
Who were the greatest influences on the choice of the processor for your current project?

Figure 10

Have you personally been involved in embedded development projects that have been partially or completely outsourced?

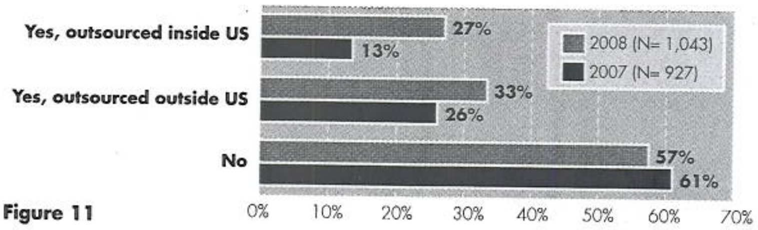

Figure 11

related to laying people off and having someone in another part of the world do the same job for less money, oftentimes a lot less money. Whether that's true or not is certainly debatable, and we've run our share of articles on this topic in the past.

You'll see in **Figure 11** that the number of project outsourced both inside and outside the U.S. is growing. But note that while more projects are going outside the U.S., the differential between those staying and those going is much smaller. The reason behind that is because the actual cost of development in countries like India, for example, is increasing.

Another response that I have a tough time justifying is that developers this year consider the "chip itself" to be more important than the ecosystem surrounding the chip (such as software, tools, and support). I constantly harp on the processor vendors how important it is to have their ecosystem in place, and that's the only real road to success. System developers seem to think otherwise.

The number of developers that don't use programmable logic in their designs stands at 52%, a relatively (and surprisingly) high number. When we asked them why, the top answers were that programmable logic is too expensive, consumes too much power, and is too hard to use. The vendors I spoke to all refuted these claims, but it appears that that message is not getting out.

Finally, it looks as if a response I was surprised by last year was not a fluke—there's no loyalty toward analog vendors. In fact, there's even less loyalty this year than last. The number of developers who will employ whatever brand meets their requirements rose from 37% to 39%. The number that has no preconceptions about the brands and will consider them all rose from 26% to 31%. And the number that will always use the same trusted brand fell from 12% to 10%. Ouch. ■

Richard Nass is the editorial director of TechInsights' Embedded Systems group. You may reach him at rnass@techinsights.com.

ADVANCED TOOLS GOOD; GOOD PRACTICES BETTER

While it's no surprise that embedded systems designers have yet again pin-pointed the meeting of deadlines as their biggest concern, the good news is that virtual platforms and graphical system-design tools are evolving rapidly to help them meet those deadline challenges. That being said, there's no substitution for good, solid firmware-generation practices to ensure a solid design.

The argument for virtual platforms from the likes of VaST or VirtuTech is pretty strong: they enable programmers to start developing code in parallel with hardware and allow reasonably accurate test and optimization before ever going to silicon. For their part, graphical systems-design environments such as National Instruments' LabView completely abstract the embedded designer from the code development and enable a system-level approach that greatly accelerates development time.

While the advantages of such tools are clear, they aren't a panacea. Virtual platforms are expensive and rely on the vendor providing accurate models of the target IC. Also, "They are just one component of firmware engineering and don't address crucially important areas like design, inspections, standards, etc.," said Jack Ganssle, embedded systems consultant. Ganssle, a regular contributor to *Embedded Systems Design* also delivers a course on this topic at each Embedded Systems Conference (catch the next one at ESC Boston on Oct. 30).

Also, code developed using LabView, whether it be C or HDL, is appreciably slower than hand-optimized code, originally up to five times slower. More recently, however, that differential has decreased dramatically. EEMBC benchmarked the LabView Microprocessor SDK and found it to be 1.05 (5%) to 2.3 (103%) times slower. Such differential reductions are set to continue as a result of tweaks such as in-line code capability where hand-generated code can be inserted in critical paths to overcome bottlenecks. In addition, the recently announced LabView 8.6 adds Component Level IP (CLIP) capability that allows the insertion of third-party IP for LabView FPGA.

Despite the improvements, the overhead remains, so nothing as yet quite replaces good firmware development practices, said Ganssle.

—Patrick Mannion

Patrick Mannion is editorial director for TechOnline, Embedded.com's sister site. You may reach him at pmannion@techinsights.com.