

# Change Is Good

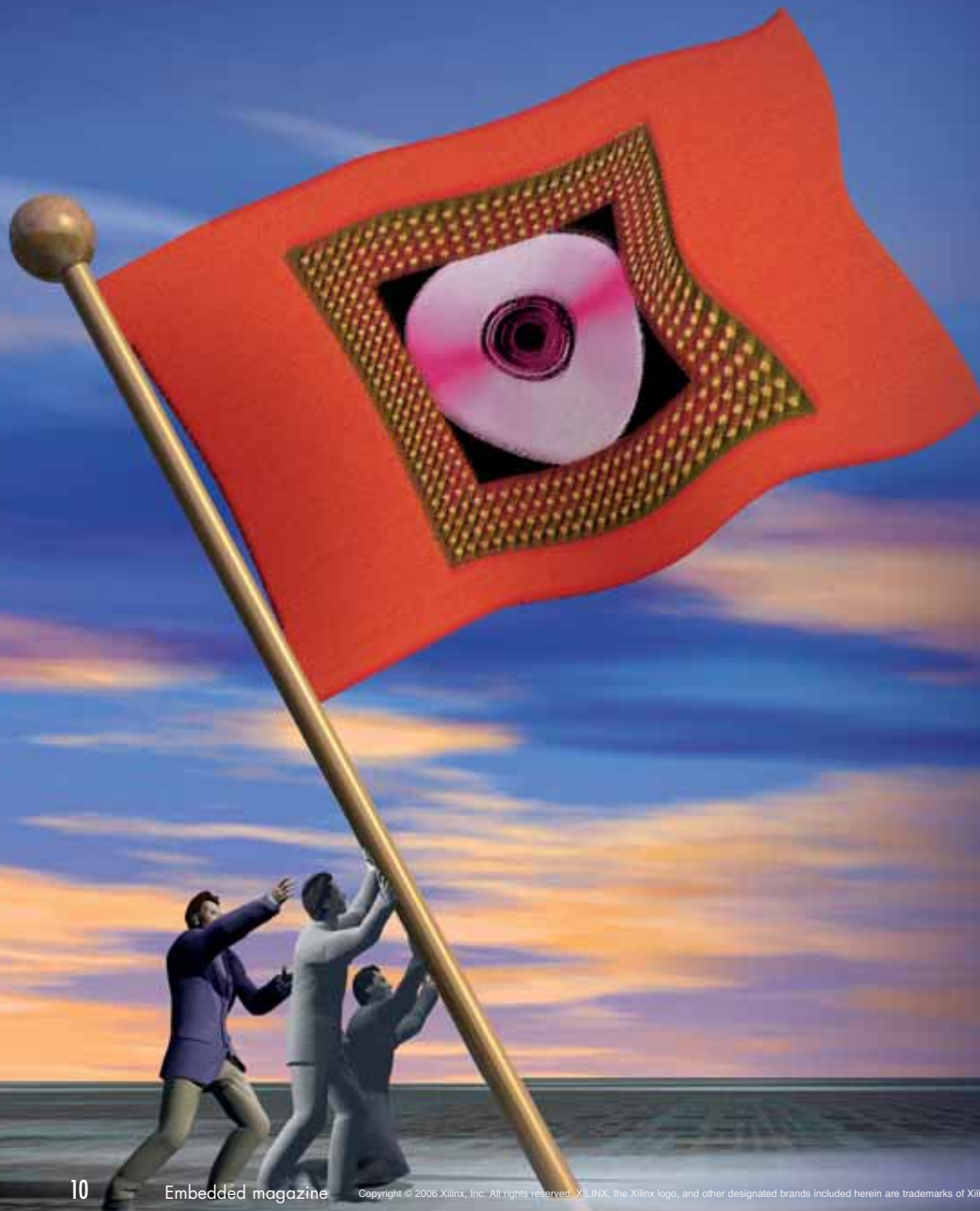
Hardware designers and software designers can't often agree, but there is a middle ground that both might enjoy.

by Jim Turley  
Editor in Chief, Embedded Systems Design  
CMP Media LLC  
[jim@jimturley.com](mailto:jim@jimturley.com)

If you shout “microprocessor” in a crowded theatre, most people will think “Pentium.” Intel’s famous little chip has captured the public imagination to the point where many people think 90% of all the chips made come streaming out of Intel’s factories.

Nothing could be further from the truth. The fact is, Pentium accounts for less than 2% of all of the microprocessors made and sold throughout the world. The lions’ share – that other 98% – are processors embedded into everyday appliances, automobiles, cell phones, washers, dryers, DVD players, video games, and a million other “invisible computers” all around us. PCs are a statistically insignificant part of the larger world – and Pentium sales are a rounding error.

Take heart, embedded developers, for though you may toil in obscurity, your deeds are great, your creations mighty, and your number legion. With few exceptions, most engineers are embedded systems developers. We’re the rule, not the exception.



## Processors As Simulators

What is exceptional is the number of different ways we approach a problem. All PCs look pretty much the same, but there's no such thing as a typical embedded system. They're all different. We don't standardize on one operating system, one processor (or even processor family), or one power supply, package, or peripheral mix. Among 32-bit processors alone there are more than 100 different chips available from more than a dozen different suppliers, each one with happy customers designing systems around them. Hardly a homogenous group, are we?

There's even a school of thought that microprocessors themselves are a mistake – a technical dead-end. The theory goes that microprocessors merely simulate physical functions (addition, subtraction, FFT analysis), rather than performing the function directly. Decoding and executing instructions, handling interrupts, and calculating branches is all just overhead. A close look at any modern processor chip would seem to bear this theory out: only about 15% of the chip's transistors do any actual work. The rest are dedicated to cracking opcodes, handling flag bits, routing buses, managing caches, and other effluvia necessary to make the hardware do what the software tells it to do.

The only reason processors were ever invented in the first place (so the thinking goes) is because they were more malleable than “real” hardware. You could change your code over time – but you couldn't change your hardware.

But that isn't true any more.

Following this line of reasoning, the right approach is to do away with processors and software altogether and implement your functions directly in hardware. Forget that 85% of processor overhead logic and get right down to the nuts and bolts. Make every one of those little transistors work for a living. And hey, if you change your mind, you can change your hardware – if it's programmable.

## The Malleable Engineer

So now we're faced with the proverbial (and overused) paradigm shift. We can toss out everything we know about programming, operating systems, software, real-time code, compilers, boot loaders, and bit-twiddling and go straight to hard-wired hardware implementations.

Or not.

Maybe we like programming. There's something about software design that appeals to the inner artist in us. It's a whole different way of thinking compared to hardware design, at least for a lot of engineers. Software is like poetry; hardware is



like architecture. There's plenty of bad poetry because anyone can do it, but you don't see people tossing up buildings just to see if they stand. Programming requires much less discipline and training than hardware engineering. That's why there are so many programmers in the world.

This is a good thing. Really. The easier it is to enter the engineering profession, the more (and better) engineers we're likely to have. And since hardware- and software-design mindsets are different, we get to draw from a bigger cross section of the populace. Variety is good.

More to the point, it's no longer an either/or decision. The two disciplines are not mutually exclusive; engineering is not a zero-sum game. We don't have to come down firmly on the side of hardware or software; we can straddle the middle ground as it suits us. When your hardware is programmable, you can choose to “program” it or “design” it using traditional circuitry methods. Take your pick. Let whimsy or convention be your guide.

Engineers, like most craftsmen, place great stock in their tools. A recent survey revealed that most developers choose their tools (compiler, logic analyzer, IDE) first and the “platform” they work on second. For example, they let their choice of compiler determine their choice of processor, not vice versa. The hardware – a microprocessor, generally – is treated as a canvas or work piece on which they ply their trade. This comes as a bit of a blow to some of the more traditional microprocessor makers, who'd always assumed that the world revolved around their instruction set.

The takeaway from this part of the survey was that keeping developers in their comfort zone is paramount. Engineers don't like to modify their skills or habits to accommodate someone else's hardware. Instead, the hardware should adapt to them. In the best case, the hardware should even adapt to a code jockey one day and a circuit snob the next. Different tools for different approaches, but with one goal in mind: to create a great design within time and budget (and power, and heat, and pinout, and cost, and performance) constraints.

There hasn't been anything to accommodate this flexibility until pretty recently. Hardware was hardware; code was code. But with “soft processors” in FPGAs living alongside seas of gates and coprocessors, we've got the ultimate canvas for creative developers. Whether it's VHDL or C++, these new chips can be customized in whatever way suits you. They're as flexible as any software program, and as fast and efficient as “real” hardware implementations. We may finally have achieved the best of both worlds. ●●●