print | close

11 Myths About 8-Bit Microcontrollers

Electronic Design
Wayne Freeman
Mon, 2016-12-12 15:27

The venerable 8-bit microcontroller (MCU) celebrated its 44th birthday this year. Even in middle age, the scrappy, numerically challenged architecture still finds its way into many new embedded applications. Of course, the last four decades of technological progress have spawned faster, more powerful MCUs that constantly threaten to occupy the market space reserved for 8-bit machines.

Engineers who are young enough to have never coexisted with the Berlin Wall or the Soviet Union tend to forgo the efficient simplicity of 8-bit MCUs in search of the promise of infinite processing power that 32-bitters offer. But are 32-bit MCUs really the end-all, be-all for every application? Let's explore some of the very common myths and misconceptions uttered about the elder statesman of the embedded market.

Related

11 Myths About Floating-Poly Embedded Flash

What Is Inside an IoT Chip?

And the Best Micro for Beginner Learning is...

1. 8-bit is going away.

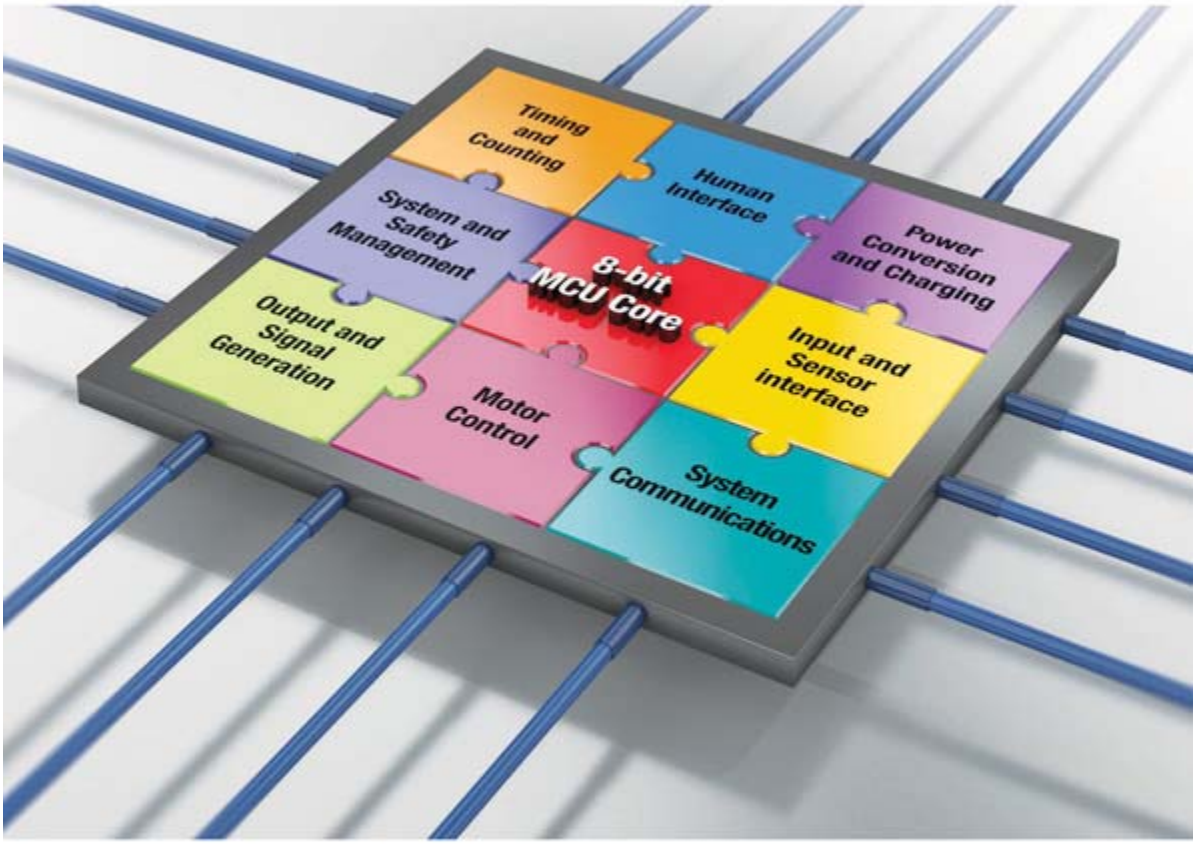
Believe it or not, this is the most common misconception we encounter in the 8-bit embedded space. If you really think about it, it's akin to saying that inexpensive, energy-efficient cars will eventually go the way of the dodo bird now that SUVs have been invented. Of course that's not going to happen... and in a similar vein, there will always be a space in the embedded world for inexpensive, energy-efficient 8-bit MCUs.

Let's try to add a bit of data to this discussion. Gartner's *2015 Microcontroller Report* lists the dollar value of both 8- and 32-bit annual sales in the ~$6B range— roughly equivalent to each other. Given the differences in average transaction price, the math tells us that there were three 8-bit MCUs designed into embedded systems for every one 32-bit MCU in 2015. Going away? Not anytime soon.

2. There's nothing new in 8-bit. No innovation.

It's easy to make the assumption that MCU manufacturers spend all of their R&D budgets on newer 32-bit product lines, while letting their aging 8-bit MCU portfolios languish with *maybe* a die shrink or two to reduce production costs. The truth is that all 8-bit MCUs have evolved over the years out of the necessity to meet the requirements of the embedded marketplace. Some MCU manufacturers do this by using technologies developed for their 32-bit portfolios to update their 8-bit MCUs. Others, like Microchip, focus on the needs of their customers to bring true innovation to the 8-bit space.

example, Microchip's line of 8-bit PIC and AVR MCUs feature core independent peripherals (CIPs) ıtelligent peripherals that operate without CPU intervention and can communicate with each other *(Fig. 1)*—that help increase system performance and responsiveness while reducing power consumption. Combined with modern quick-start development tools (such as MPLAB Code Configurator and Atmel START), these 8-bit innovations enable designers to go from prototype to production-ready in a matter of months.



3. 8-bitters are really difficult to program with C and other high-level languages.

Yes, it's true that 8-bit architectures pre-date the widespread use of high-level programming languages in embedded design. In fact, a lot of old-school engineers still refuse to install a newfangled C-compiler into their embedded environment. While that's entirely okay—we don't judge—it would behoove us to note that most 8-bit MCUs sold now make it pretty easy to develop code with a modern C compiler.

Microchip's modern PIC16 "F1" MCUs have several instructions entirely dedicated to eliminating any issues dealing with its paged/banked address space. If that's not enough, both PIC18 and AVR architectures feature massive linear address spaces that are tailor-made for high-level language compilers. Of course, modern integrated development environments (IDEs) have made all of these concerns a non-issue.

4. You can only use 8-bit MCUs for simple applications.

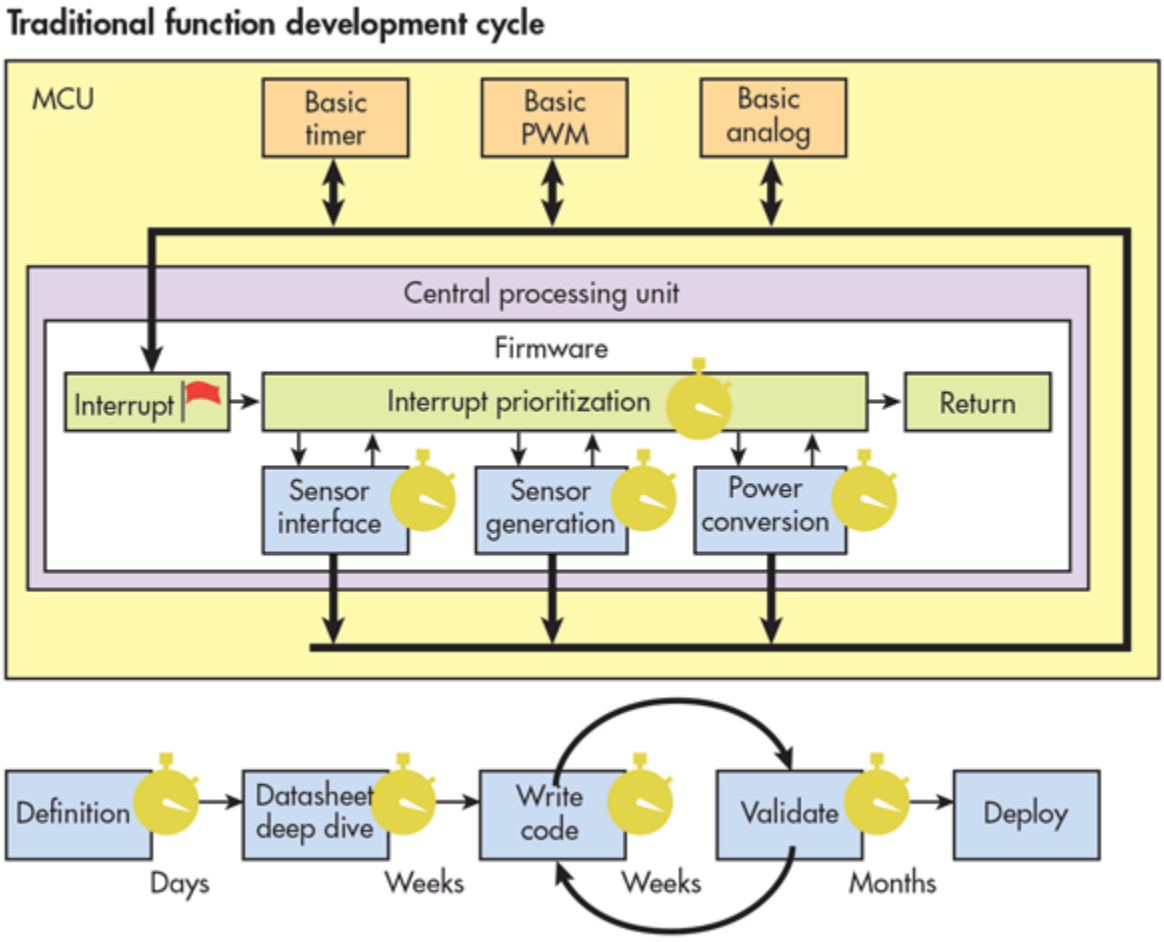
This depends on how you define the term "simple application." The vast majority of embedded systems don't require massive amounts of compute power and an embedded OS just to keep everything in sync. The most common embedded systems involve some type of sensing, power conversion, closed-loop control, or low-data-rate communication activity. This puts them firmly in the "simple application" category. With the right on-chip peripherals, these tasks can be handled easily by an inexpensive MCU.

that end, Microchip has developed more than 20 CIPs to autonomously handle the most common embedded ‹s, leaving the CPU to perform a supervisory role. Using CIPs makes an application's response more deterministic, in addition to being far simpler to implement (no pesky interrupt service routines to code and debug). In this case, simple is a good thing.

5: 8-bit isn't powerful enough for IoT designs.

It's easy to get caught up in the notion that every Internet of Things (IoT) application looks like a smart watch, wireless audio node, or some other equally complex embedded system. The reality is that the "things" comprising IoT applications perform pretty simple tasks. They're touch-sensitive switches, sensors, wireless light bulbs, and networked garage-door openers; it's the button in your laundry room that automatically reorders the detergent when you press it.

These systems take a measurement. They send data to the internet through your home's router. They control a simple switching power supply. They wake up when "nudged" by the network. They consume very little power while doing their job. All of these are tasks that newer 8-bit MCUs are specifically designed to perform. If your next IoT design doesn't feature an 8-bitter, you may be missing an opportunity to optimize your application.



6. 8-bitters aren't fast enough to respond to critical system events.

Remember this: Response time to critical system events is all about latency and determinism.

It's no secret that almost every 32-bit MCU running at full speed will outperform an 8-bit MCU running at its maximum clock speed. Most applications, however, don't have the power budgets to support any MCU running

ull speed 100% of the time. So, a common design practice is to utilize the CPU's sleep mode to lower power sumption when it's not needed, and to wake it up with a hardware interrupt in the event that it's needed again. With this technique, you've just introduced two of the biggest barriers to on-time application performance: variable startup time and interrupt latency. Both of these must be carefully considered and mitigated when coding an application, otherwise the results can be disastrous.

MCUs with CIPs allow designers to implement the most timing-sensitive parts of their application in fixed-function, low-power and always-on hardware. Doing so provides a consistent, low-latency response to critical events with extremely low power consumption.

7. 32-bit MCUs are more power-efficient than 8-bit MCUs.

Just as everyone knows that 32-bit MCUs are the fastest kids on the embedded block, they also understand that faster 32-bit speed is matched by higher 32-bit power consumption. No argument here, but some engineers believe that they can use all of that speed to "cheat the system" a bit. Basically, by getting your work done faster, you can put the CPU in sleep mode for longer periods of time. Thus, 32-bit MCUs are more power-efficient than 8-bit MCUs, right? Wrong.

Even if you ran roughly equivalent software routines on both architectures, the much lower run-mode power consumption of the 8-bit MCU would guarantee longer battery life for the tiny powerhouse. Add to that the fact that most modern 8-bit MCUs have a better balance of peripheral features than similarly priced 32-bit devices. This allows the "less powerful" MCUs to handle more tasks in hardware, supporting longer CPU sleep-mode times. Several common applications require nearly *zero* CPU uptime from newer PIC and AVR MCUs—advantage: 8-bit.

8. 32-bitters are the same price with twice the performance.

If you've stayed with us this far, then you very likely understand that performance in embedded applications is measured in more than just raw computing power. The lowest-cost 32-bit MCUs tend to have great CPU performance and lots of inexpensive digital peripherals, but are usually lacking in the on-chip analog features often needed to implement many embedded systems.

Microchip's 8-bit MCUs feature intelligent analog peripherals that can automate signal-analysis tasks, provide compensation info to digital pulse-width modulators (PWMs), and provide auto-shutdown capability without CPU intervention. These 8-bit MCUs can help significantly reduce external component count, saving a ton of money in the process. The best advice is to look at the actual needs of the entire application, and choose the MCU that will complete the task with the lowest system cost. You'll never go wrong right-sizing your system.

9. 8-bit isn't a future-proof solution.

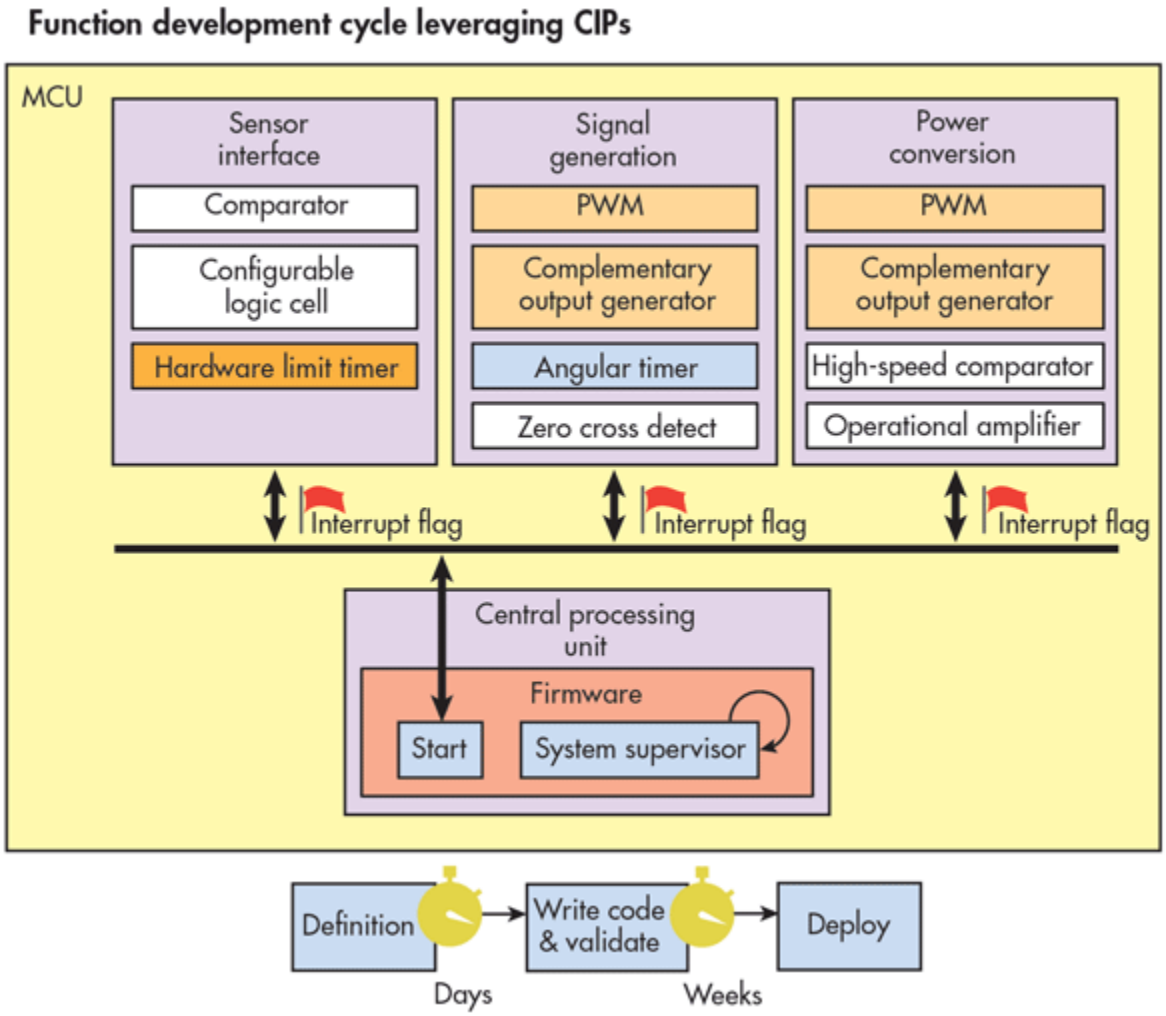
Selecting an MCU with an eye toward unpredictable market trends in the distant future is never guaranteed to be successful, but will almost certainly come with unintended consequences. It's impossible to determine what the future will hold. If the market shifts to an entirely different set of functional requirements, a complete system redesign will be necessary regardless of the 8-, 16- or 32-bit MCU being used.

Select the "best-fit" architecture for any project based on needs and requirements that can be identified at the design's onset. Doing so will help keep both hardware- and software-development costs to a minimum. Often, an 8-bit PIC or AVR MCU will help lower overall system hardware cost, and the functions enabled in on-chip hardware can massively reduce software efforts over the life of a design.

10. 32-bit MCUs reduce software-development time.

iting code is easy. Writing code *that works* is another endeavor entirely. People who subscribe to this myth ally underestimate the amount of time and effort required to test, debug, and validate their code in a working application. Either that or they're all coding geniuses. A day's worth of coding can lead to months of validation *(Fig. 2)*.

Applications that use 32-bit MCUs tend to be software-centric. That is, all necessary functionality is implemented in software routines—so there's no efficient way to escape the validation cycle. Both 8-bit PIC and AVR MCUs are designed to more efficiently use their peripherals in order to reduce the number of lines of code needed to implement common functions. This shortens the development process, since coding is a snap and the hardware's functionality has been validated from the factory *(Fig. 3)*.



To sum it up, the best way to reduce software-development time is to write fewer lines of code in the first place.

11. There's no upward migration path for 8-bit.

Upward migration depends more on your development environment than any single MCU or piece of hardware. If migration path is a key concern, then your best bet is to source your MCUs from a "full line" manufacturer who offers 8-, 16- and 32-bit MCUs supported by a consistent development ecosystem. These days, it's the development environment, not the hardware, which determines the ease of migration from one MCU to another. A well-crafted IDE, such as Microchip's MPLAB X or Atmel START, enables users to implement their ideas on any MCU in its broad portfolio.

ou're designing the next Linux-based wrist-worn supercomputer, then it makes perfect sense to begin and your MCU search with the latest and greatest 32-bit devices. But if you're like the rest of us adding intelligence, greater control, and determinism to common everyday systems, then you'd be remiss if you didn't add newer 8-bit MCU offerings to your list. You'll find the lowest power consumption, incredibly useful peripherals, and a super easy development experience. What's not to like about that?

Source URL: http://electronicdesign.com/microcontrollers/11-myths-about-8-bit-microcontrollers