A Byte is a Byte

Murat Can Cagri (Armelsan Defence)

13.02.25

Summary

“You are never more than six feet away from an 8051”.

Non 8-bit systems are not rare, esoteric nor they are a relic of forgotten times, despite the claims made in the original paper, which were then retracted in a future revision in favour of “they exist, but they don’t have modern C++”. If we are still considering C++11 *the* modern C++ revision, this new argument also falls short as these systems have do have support for modern C++. But it is indeed true that embedded systems are falling behind and many still don’t have support for *postmodern* C++, which we will discuss later in this paper.

Vendor/Family	Byte Width	Compilers/Tools
C166/ST10 DSPs	16-bit	- VX Viper C++11- Tasking Classic C166 C++98- Keil C166 C++98
Texas Instruments C2x Series DSPs	16-bit	- TI C2000-CGT C++11 Compiler
NXP DSP56k DSPs	16-bit	- Tasking DSP56k C++98 Compiler
Analog Devices SHARC Family DSPs	16/24-bit	- CrossCore Embedded Studio C++11

16-bitsillion Devices Run C++

C166/ST10 series are a good example with more than 500 million units sold since 2007, and are still widely available in production. With their VX Viper Compiler supporting C++11 with 16-bit bytes, alongsiVde 4-bit addressing for C166SV2 targets, they offer support for XE166 and allow many of us to develop for these systems. There is also a legacy C++98 compiler from Keil for this target. This is just a single family of MCUs targetting automotive with more than 500 million units, and it cannot be considered a rare product, or as a relic.

Another example is Texas Instruments C2x series, a 16-bit byte widely used DSP in motor controllers with excellent processing characteristics. This particular model is used by us right now. The compiler for this DSP is supplied by Texas Instruments, and has C++11 support. We will dive deeper into the tooling in embedded systems later in this paper. It is worth mentioning that, moving forward with P3477, Matlab will have support for this platform, but C++ will not.

Regarding obsolete systems; DSP56k is an example worth mentioning, originally from Motorola, then Freescale, and currently from NXP. This particular DSP is targeting Digital Audio Processing with a 24-bit byte, and is now obsolete. The reason to mention an obsolete DSP, is to show that although a specific model can die, the architecture still lives on. In this case DSP56300 of the same architecture is still in production. we have a lot of devices out there, and dropping support for them is not feasible, the same way it is not feasible to rewrite entire codebases in some other language. The fact of the matter is that these systems work and they have no reason to stop working just because we decided to do so.

Yet another similar architecture, is the SHARC family of DSPs from Analog Devices, another Digital Audio Processor, with their 214xx family having 24/16-bit bytes. The C++11 compiler for this platform has also support for user configurable CHAR_BIT.

Overall, there is about a billion of these devices from reputable companies, and these devices are not merely maintained –although that should be enough reason to support them– but also being produced and developed for actively. The digital design engineers who are responsible for these architectural choices, have strong reasons to do so. 24-bit bytes are natural when you are working with anything audio related, or working with hydrophones. These design choices are made by experts who are responsible for designing a performant DSP with minimal silicon area and high-efficiency, and they have no reason to mandate an 8-bit byte into their design at the cost yield, performance and efficiency.

Even if the case made by the original author would have been true, there is historical precedent that shows how rigid assumptions fail. This paper will discuss why overstandardisation should be avoided in detail in the following section.

Overstandardisation, Underrepresentation

The process of standardising C++ is lengthy, complex and detail oriented , where we get actual feedback a lot later in the pipeline. This is especially problematic for embedded developers, who are underrepresented when important decisions regarding their domain is being made. Due to its nature, embedded development also has similar constraints to the ISO C++ process, where developers must act carefully. It is not easy to deploy a Day-0 patch, when your product is deployed in tens of thousands at a time in a remote location. Sending technician with a JTAG to the James Webb Telescope is not feasible, so careful design choices must be made.

The natural consequence of this delayed-feedback, results in new features being dissected ever more carefully. Tread lightly and stick to known norms that are proven to work robustly, so is to not brick tens of thousands of devices. This type of development suffers greatly, when your tooling also has a similar process. It takes time and effort to see the consequences of certain actions. This is the reason Deprecation of Volatile took years to trickle down to tooling, but when it actually did, things did not go well.

By no means, embedded development is a niche field. It just happens that the actual feedback from developers trickle down a lot later to the people who are responsible for the tools they use. A direct result of that is the underrepresentation, where hasty decisions come to pass hastily, but reverted and/or fixed in a lengthy time frame with an equality lengthy process. This causes a lot of trouble, especially for the embedded development field.

The now changed main argument of P3477R1 is that the vendors lack behind, and this is enough reason to drop support for possibly billions of devices. An important thing to see is that the vendors have no reason to support C++ if C++ has no intention to support their respective platforms. The vendors don't support C++, because C++ doesn't support vendors is a circular logic.

Furthermore, there is another argument: If, these devices do in fact exist –and they do–, they can just use C, but there are also claims where "C should follow P3477 soon", which means we are not only dropping support, we are killing these devices. This is a serious case of underrepresentation of the interests of embedded developers. C++ is a language for everyone, and every hardware.

Another consequence of this lagging feedback materialised itself in the form of an ever harder to implement language. It is not easy to write a compiler, and it isn't getting any easier. It took years for C++11 to catch up, and the compilers trickled into embedded development even later, nevertheless, lagging behind does not mean stagnation, but dropping support does. There is no reason to expect embedded developers to use legacy compilers forever if there is no light at the end of the tunnel. People will drop C++, before C++ drops them. The recent papers regarding freestanding library has been well received, and good progress is being made, it is essential that we keep that direction. Similar papers to P3477 caused a lot of trouble; most notoriously Deprecation of Volatile. Are we willing to do the same and go through this same process once again? The argument that these platforms don't support modern C++ is a self-fulfilling prophecy, caused by itself. The proposed change is quite different than enforcing 2s complement, and it will have actual impact.

The Design of C++

As a direct result of an ever harder to implement C++, we are now in the process of overstandardising a language to make it feasible to implement for compilers. Such goals must be supported, but under different rationales. There is historical precedent that show how our rigid assumptions of the hardware fails, whether it would be the dominance of x86, or when 32-bit architectures were considered the norm. If we were to standardise an int as 2-bytes back in the day, we would be in way more trouble now. Some things are not meant to be strictly defined, not only for the sake of existing architectures, but for architectures to come. We have a model of computation defined by our parametric non-deterministic abstract machine, as opposed to languages with a virtual machine. This is there to allow C++ to run everywhere, now and forever. We must protect our ability to port Doom to an abacus by all costs. The reason our standard doesn't define the stack, is because ever so often there comes a platform where such notion doesn't make much sense, such with the case of WASM.

Our process should be driven by domain experts where we evolve the language around our design philosophies and evolution policies. We don't break code, and we do not leave room for a lower level language. Dropping support for devices and expecting people to write C for their needs, is the exact meaning of leaving room for a lower-level language. We are supposed to be the language for everyone.

Who P3477 (There are exactly 8 bits in a byte) is For

One of main motivations behind P3477 is that it effectively changes nothing. If it doesn't change anything, why are we talking about it? If it is in fact a needed change, how does it not change anything? People who don't need to think about the number of a bits in a byte, still won't have to, but people who need this, won't be able to use the language, for seemingly no gain. Those who need non 8-bit byte support already know what they are doing, and the standard has no reason to restrict the flexibility just because most developers never encounter this problem, and the argument that this is a confusing feature is unwarranted, given that we have std::launder and std::hardware_destructive_interference_size, which might be confusing for an everyday developer, but an important feature of the language nonetheless.

With the proposed change, people who write code for 8-bit byte systems will continue to do so as they did. If it is not a concern for the specific platform people are using C++ to develop for, then it doesn't warrant a solution. The existence of a language construct does not require a developer to actively think about it nor it requires the said construct to be removed if it is domain specific.

Standardising for convenience ignores the hardware realities driving these decisions. It is unclear what is the positive impact of the proposed change is, and whose convenience we are targetting for.

References

P3477R1: There are exactly 8 bits in a byte

C166 Series Product Brochure