De-deprecating volatile compound operations

Paul M. Bendixen

2021-02-12

1 Introduction

The publication of the ISO C++20 standard included the deprecation of many functionalities of the volatile keyword.

This was due to a paper [Bastien, 2019] which slated these for removal because of unfortunate usages of volatile qualified usages in threading code. The reasoning behind was presented in an earlier version of the paper[Bastien, 2018].

While the reasoning behind might be sound, unfortunately the deprecation of one particular part of the volatile functionality, bitwise compound assignments, caused some response from the embedded community[Ooijen, 2020].

The purpose of this proposal is to revert a small part of [Bastien, 2019], it is the hope of the author that a compromise that favours all parties can be reached.

2 Problem statement

2.1 Background

One of the great advantages of C++ is its closeness to the machine on which it operates. This enables C++ to be used in very constrained devices such as microcontrollers without any operating system. These systems often operate by manipulating memory mapped registers often only touching a single bit.

While there are multiple ways to manipulate single bits in a variable the most idiomatic way is something along the lines of the following:

```
// In vendor supplied hardware abstraction layer
struct ADC {
        volatile uint8_t CTRL;
        volatile uint8_t VALUE;
        ...
};
#define ADC_CTRL_ENABLE (1 << 3)


// in user code
ADC1->CTRL |= ADC_CTRL_ENABLE; // Start the ADC
...
ADC1->CTRL &= ~ADC_CTRL_ENABLE; // Stop the ADC
```

This is further amplified by the fact that vendors may supply macros for setting or clearing bits utilizing this idiom, or may use it in provided code generators.

It is clear from the previous example that this idiomatic usage of compound operations clashes with the deprecation of compound operations on volatile. The argument against compound operations on volatile variables is that it leads the programmer to believe that the operation itself becomes compounded and therefore atomic. This is of course false (even though architectures where this is the case could be imagined).

Now since there must always be a read, modify, write cycle it is possible that an interrupt would happen in between the read and the write, however most code that uses this idiom needs to take care of this by being right by construction i.e. not bit-twiddeling variables that are used in the interrupt service routines (ISRs). While there is probably some code found in the wild where this would indicate a problem, since this is so common in all these cases the construct would only be replaced by the equivalent non-compound statement, giving no benefit at all.

2.2 Scope

So what is the impact of deprecating compound operations on volatile? A search of some of the more widely used embedded libraries show that while it is not massive it is in some way in all codebases for embedded systems available today.

The numbers below are done by using UNIX grep[1] to find usages of either |= or &= in the code bases of the respective libraries. While this method isn't foolproof it does give an indicator of the problem.

Library	compound operations
Silabs Gecko SDK	293
AVR libc 2.0	205
Raspberry pi Pico SDK	13

Table 1: Occurrences of compound operations on variables typically associated with volatile

This illustrates the major point, not only is there a lot of code already written out there *working as expected* but the usage of this idiom also prevents adopters of C++ to use the C libraries provided with their toolchains.

The idea that businesses will be willing to spend the effort to update their codebases that become defunct because of this changes seems unlikely, rather it seems likely that they will mandate using no newer versions.

2.3 Didn't we just go over this?

While the proposal to deprecate was heard in committee meetings, the main focus was on problems arising with multi-threaded code (SG1) and with EWG,

[1]The search was done using the command

```
grep -re "[A-Z0_9]\+\(\[.*\]\)*[ ]\+[&|]=" -include \*.h .
```

neither of these groups can be expected to be familiar with the inner workings on microcontrollers.

3 Possible solutions

3.1 The simple

The simplest possible change that could possibly work would be to remove the text added to paragraph [expr.ass] point 6 as this would allow compound statements on volatile varaiables.

[Note: Would [expr.ass] 5 of the draft standard remove the posibility to assign to any volatile variable? In that case we need to remove that one too — end note]

3.2 The compromise

As the compound operations are mainly used to flip bits, a compromise could be to only de-deprecate the use of binary compound operations (`|= &= ^=`) as these are the ones that are useful for this purpose.

This would require new wording to be put in.

4 Thanks

Thanks to Jens Maurer for the suggestion on the compromise solution.

Bibliography

[Bastien, 2018] Bastien, J. (2018). P1152r0 deprecating volatile. Technical Report P1152R0, ISO. Retrieved at `wg21.link/P1152R0`.

[Bastien, 2019] Bastien, J. (2019). P1152r0 deprecating volatile. Technical Report P1152R4, ISO. Retrieved at `wg21.link/P1152R4`.

[Ooijen, 2020] Ooijen, W. V. (2020). Compound assignment to volatile must be un-deprecated.