

Why Should Auto-Generated C be Treated any Differently from Hand-Coded C?

Steve Montgomery

Ricardo UK Limited

Copyright © 2008 SAE International

ABSTRACT

Many embedded software developers producing C code for automotive applications will be familiar with the MISRA C guidelines. These provide practical guidance in avoiding some of the common pitfalls associated with the C language. They were originally meant for automotive developers, but are now increasingly used in other critical sectors such as aerospace and medical devices. The commonly-held view, expressed in the MISRA C guidelines, is that all C code, whether hand-written or automatically generated, should be subjected to exactly the same verification and validation activities. However, as modeling language and automatic code generation techniques have evolved over the 10 years since the first edition of MISRA C, this advice may no longer be so relevant. This paper looks at some of the issues related to verification and validation of automatically-generated C code and identifies the situations in which the authors believe that a different approach is warranted.

INTRODUCTION

Many automotive suppliers and manufacturers are using automatic code generation techniques as part of their development process for series production automotive embedded software. Software designs are developed using a graphical modeling package and then translated by an automatic code generator into code which is usually (but not always) in the C language. There are several attractive features with this approach when compared with hand-coding software in a language such as C:

- Modeling languages tend to be more expressive than the imperative languages that are commonly used in automotive engineering
- Software designs produced using a modeling package can be executed and tested in simulation
- Automatic conversion of a design into code reduces the probability of an error being introduced at this phase of development

All of these attractive features of automatic code generation deliver benefits in terms of reduced development effort and cost.

The expressive power of a language is related to the ease with which it can be used to describe an algorithm. Assembly languages, which were once used in some of the earliest automotive embedded software systems, have very little expressive power: each statement in the source program corresponds to a single machine instruction. High-level imperative languages used in automotive control software, such as C, have a higher expressive power: a single C statement might correspond to tens of machine instructions. Modeling languages have an even higher expressive power, in that a single block might correspond to many C statements and therefore around 100 machine instructions. Studies suggest that the number of lines of debugged code per day that a software developer can produce is roughly independent of the language being used. All other things being equal, productivity is therefore likely to increase if a highly expressive language is used. Of course, all other things are not equal, and there are prices to be paid, not only in terms of modeling tool investment, but also in terms of reduced flexibility, increased code size and reduced execution speed. However, given the complexity of current and future automotive software systems, any increases in project cost due to use of automatic code generation are likely to be repaid many times over in increased productivity.

The ability to perform tests on an executable design provides the opportunity to confirm that the design is sound before coding commences. It is widely accepted that the cost of rectifying a software design defect during the design phase is far cheaper than rectifying it in a later design phase, or worse, once the software has entered production.

The automatic conversion of design into source code may not necessarily deliver a significant reduction in the overall effort required to code the design, because the investment needed to set up the process can be high. The return on this investment only becomes significant if the design changes many times during the project. However, regardless of whether the automatic code generation process reduces overall coding effort, it should significantly reduce the possibility of certain types of defect being introduced during coding. This means

that it should be possible to reduce the amount of effort spent trying to find defects in the code.

A key question, for which there does not appear to be a universally agreed upon answer, is how to reduce the effort required to find coding defects. The MISRA C Guidelines recommend that all of the code verification steps that are applied to hand-written code should equally be applied to automatically generated code. But if this guidance were to be followed precisely, some of the potential benefits of automatic code generation would appear to be lost.

This paper reviews the three code verification activities that are typically employed in automotive software engineering processes and assesses the relevance of each to automatically generated code. It also provides a more in-depth treatment of the applicability of the MISRA C language subset to automatically generated code. It identifies potential pitfalls when applying the full MISRA C guidelines to automatically generated code and introduces a reduced subset, MISRA AC AGC, that is specifically aimed at automatically generated code.

USE OF MODELING PACKAGES: DESIGNING VERSUS PROGRAMMING

Before reviewing code verification activities that might be applied to automatically generated code, it is instructive to consider the role of the modeling language in the development process.

Automotive embedded software engineering processes have evolved from using low-level programming languages (assembly languages) to using high-level programming languages such as C. This reflects a general trend in programming language development and application, albeit occurring somewhat later than in other domains, such as business application development, primarily because of the desire for fine control of the processor coupled with highly efficient code.

When use of C in automotive development was in its infancy, some developers were somewhat mistrustful of the C compilers used to translate their programs, and they reviewed the assembly language output of the compiler as a matter of routine. However, as confidence in the C language and its compilers grew, this practice ceased and developers began to think more naturally of programming in C.

It is this ability to focus on the high-level programming language rather than the machine instructions that is the source of the power of high-level languages. It is made possible because the C programming language provides an abstraction of the processor on which the program will run. This abstraction is defined in an international standard, ISO 9899 [1], and has been essentially stable for nearly 20 years. Any competent programmer who is familiar with the C language standard can use this abstraction to write programs that will operate correctly,

regardless of the type of processor on which the program is executed. Of course, in an embedded application there will be parts of the program that need to handle processor-specific features, such as input and output. While these are typically encapsulated, since they rely on processor-specific features of the C language, they still use the same abstract programming model.

At first sight, it might appear that similar considerations apply to the relationship between modeling languages and the C code produced by an automatic code generator:

- a modeling language is a higher-level language than C, just as C is a higher-level language than assembly language
- some developers mistrust automatic code generators and conduct reviews of the C code generated by them

However, there is an important distinction between most modeling languages and the C programming language. The C language abstraction is reasonably well-defined and stable: there are areas where the language definition is less than perfect, but these areas are well documented, either in the ISO C standard itself or in 3rd party documents such as MISRA C [2]. By contrast, most of the modeling packages that are popular with automotive control system developers do not have an internationally-agreed upon standard. The programming abstraction is controlled by a commercial organization, is usually not well documented, and it can change as new versions of modeling packages are released.

A modeling language that cannot provide sufficient stability and detail relating to its abstraction cannot be considered suitable as a programming language. This is especially true given the heavy reliance on automotive software systems for reliable and safe vehicle operation. It therefore follows that such weakly-defined modeling languages cannot be used to generate automotive software without the need for additional development process steps to verify the code. In fact, this type of modeling language is being used as a *design* modeling tool. The fact that the design can be converted into C code automatically may save some programming effort, but it does not mean that the design model is itself a program.

Some modeling languages do provide the level of semantic definition that would be expected of a programming language. The developers of modeling packages that use these languages have typically invested significant effort in demonstrating the correctness of their automatic code generators. In these cases, developers can truly be considered to be programming in the modeling language, and the corresponding automatic code generator can be considered as a compiler.