Qualifying a C Library

Dealing with the conflicting worldviews of the ISO 26262 and ISO C standards

Gerard Vink

Functional safety and cybersecurity standards treat the qualification of tools and libraries as independent subjects. This independence is inconsistent with the perspective of the ISO C standard which serves as the foundation for implementing compiler toolsets and their libraries. This fundamental difference poses difficulties when it comes to qualifying a compiler and the libraries associated with it.

Introduction

To achieve conformance with the ISO 26262 functional safety standard, it is required to qualify software components, including the libraries that are part of a compiler toolset, before they can be integrated into automotive software systems.

The implementation of a C compiler and its associated libraries heavily relies on ISO/IEC 9899, commonly referred to as ISO C. This standard delineates several critical aspects:

- It defines the characteristics of environments used to translate and execute C programs, which form the basis for what's known as "startup code."
- It specifies the syntax, constraints, and semantics of the C language, which are essential for building the compiler and its executable file, as well as the runtime libraries.
- It covers the library facilities, describing what's inside header files and how C library functions should behave.

The above topics are addressed in separate clauses of the ISO C standard but with many references between the clauses, indicating interdependencies between the startup code, the compiler executable, the runtime libraries, the header files, and the C library. Consequently, it's not immediately clear whether the requirements for tool qualification and/or library qualification apply to a specific part of the compiler toolset, and whether changes made to one part of the toolset might impact the qualification of other parts.

Tool Qualification

Ensuring the qualification of a compiler tool for functional safety (FuSa) is a well-established practice, which is described in part 8, chapter 11, of the ISO 26262 standard. The qualification process aims to provide evidence that the software tool is suitable for use in the development of FuSa-related software.

There are four primary methods outlined in ISO 26262 to qualify a software tool:

- *Increased Confidence from Use*: This method involves gaining confidence in the tool through its extensive use in relevant applications.
- *Evaluation of the Development Process*: This entails assessing the development process of the software tool itself.
- *Validation of the Software Tool*: This method verifies that the software tool meets specified requirements for its intended purpose.
- *Development in Accordance with a Safety Standard*: This approach involves creating the software tool in alignment with a safety standard.

For higher Automotive Safety Integrity Levels (ASILs), typically only the last two methods are considered suitable. The industry predominantly relies on the tool validation method, which essentially means that validation measures exist to prove that the software tool fulfills its specified requirements.

Compiler suppliers employ two distinct approaches:

- *Certified Compiler Toolset*: Some compiler suppliers, for example TASKING, perform tool validation internally and engage a conformity assessment body to certify that both the tool and its associated safety documentation are suitable for their intended purpose. Customers receive a certified compiler toolset and only need to adhere to the guidelines outlined in the safety manual.
- *Certifiable Compiler Toolset with Tool Qualification Methodology*: Most compiler suppliers still offer a certifiable compiler toolset along with a tool qualification methodology that includes supporting tools and documentation. In this case, the tool qualification methodology is typically certified, but the customer is responsible for performing the tool qualification process. This involves steps such as specifying the use case to define the tool's requirements, selecting appropriate tests to verify these requirements, conducting the tests, analyzing the results, generating safety documents, and subsequently adhere to the guidelines outlined in the safety documents.

It is important to note that the latter approach has hidden costs such as the need to learn the qualification methodology and associated tooling, license the required compiler verification suites, perform the tool validation process, interact with the certifying authority and addressing issues that may arise if tests fail. This last part can be troublesome. What to do if the vendors of the compiler toolset and the compiler verification suite have different views on the interpretation of specific parts of the ISO C standard?

Software Component Qualification

The ISO 26262 FuSa standard provides two distinct methods for qualifying software components such as a compiler’s C library.

One is specifically designed for commercial off-the-shelf (COTS) software components while the other is for software components developed independently of their intended use. Both methods offer a structured approach to ensure that software libraries meet the necessary safety requirements. The method to qualify COTS components is often applied by parties that did not develop the software component.

Commercial Off-The-Shelf Software Components

Requirements for qualifying COTS software components are described in ISO 26262, part 8, chapter 12. They mandate the availability of detailed software component requirements, encompassing aspects like runtime environments and the numerical accuracy of mathematical algorithms, and require that the software remains unmodified.

Extracting such detailed requirements from a compiler toolset's user documentation is often impractical or impossible, and falling back on the requirements of the ISO C standard is also not a solution because aspects such as the accuracy of the C library mathematical functions is specified as "implementation-defined", making it difficult for entities other than the library developer to perform a reliable qualification.

If the above challenges can be resolved, the qualification effort boils down to showing a requirement coverage in accordance with ISO 26262 part 6, chapter 9 Software Unit verification[1], which requires access to the source code of the C library and the compiler run-time libraries, and a C library implementation specific test suite to satisfy the branch coverage requirement for ASILs B and C and the MC/DC coverage requirement for ASIL D.

For this qualification method ISO mandates that the software remains unmodified, so issues identified during the qualification activities cannot be fixed and must be resolved, if possible, via some workaround. This makes this qualification method unattractive.

Out-of-Context Software Components

Requirements for qualifying a Safety Element out of Context (SEooC) are described in ISO 26262, part 10, chapter 12. When considering the C library as an SEooC, the qualification process encompasses all ISO 26262 safety lifecycle steps relevant to automotive software development. This includes the development of a technical safety concept based on assumptions about how the software component will be used. Subsequently, it involves adhering to product development requirements at the software level, which encompass aspects such as applying a suitable development process, specifying safety requirements, creating software architectural designs, implementing software units, verifying these units, and defining integration procedures. In addition to ensuring that the component doesn't introduce additional hazards (safety lifecycle), the qualification process also addresses functional aspects to ensure the component performs as intended. This makes this qualification method much more expensive to apply than the previous COTS-based approach.

[1] It is debatable how to deal with the prerequisites of ISO26262-6:9.3.1. Taken literally, the prerequisites imply that entire software unit and architectural designs are required as input to the verification activities.

The advantage for the user of a C Library qualified as SEooC is that the library can be used without any re-qualification effort. The safety documentation contains the assumptions on safe use and provides guidance on how to establish the validity of the assumptions during integration of the library.

Decomposition of a Compiler Toolset

Figure 1 provides a decomposition of a compiler toolset and shows the relationship between the components. Specific qualification considerations apply to each component.

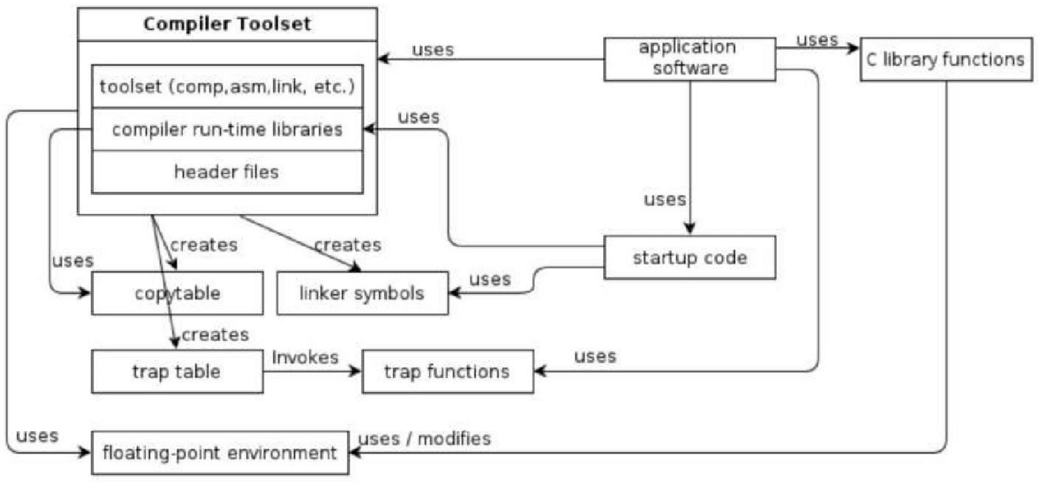

Figure 1, Relations between Application Software, QClib, and Compiler Toolset

Qualification Considerations per Toolset Component

Executable programs: A C compiler toolset contains executable programs that are typically split into a compiler, an assembler, and a linker. The tool qualification criteria from ISO 26262, part 8 chapter 11 apply to the qualification of these executable files.

Compiler Runtime Libraries: Whether a specific functionality will be built into the compiler executable or implemented in a runtime library is based on the instruction set architecture of the microcontroller and on tradeoffs between code size and the execution speed of compiler-generated code. The functions in the compiler run-time libraries are developed by the compiler developer in parallel with the development of the compiler executable. Conformity assessment bodies tend to conclude that because the runtime library functions are called by compiler-generated code and developed by the compiler developer, the tool qualification criteria apply.

Header Files: The header files (standard include files) contain macros, type definitions, and function prototypes. There is a close relationship between the implementation of the compiler's executable and the contents of the header files. For example, the macros in the header file "limits.h" must conform to the basic types as implemented in the compiler. Therefore, the tool qualification criteria should apply to header files. On the other hand, the header files are specified in the ISO C standard in the Library chapter and the header files contain prototypes of the library functions and macros and type definitions used by library functions. One could therefore also argue that the qualification criteria for software components should apply. Conformity assessment bodies take the view that if the header files are developed by the same engineers who developed the compiler, then the header files should be considered part of the compiler tool and are qualified in the context of compiler tool qualification.

Copy Table, Trap Table and Linker Symbols: The copy table, trap table, and linker symbols are data structures and symbols created during the compilation process. The contents of the data structures and the existence of certain symbols depend on the user's application being compiled and on the applied compiler/linker options settings. As such, the above entities are qualified in the context of the compiler tool qualification. These data structures and symbols are used to initialize the compiler's runtime environment, which is handled by the startup code. The connectors in Figure 1 show the dependencies between the compiler-generated entities and their use by library functions. This makes it clear that the compiler tool and compiler library qualification cannot be viewed as separate, unrelated activities.

Startup Code: The system integrator has ultimate responsibility for qualifying the overall startup software. The supplier of a qualified compiler and associated libraries is responsible for providing prequalified startup software that initializes the compiler's runtime environment, which can be easily integrated by the system integrator into the overall system startup software, without invalidating the prequalification. Initializing the compiler's runtime environment means assigning the appropriate values to the registers used by the compiler-generated code and copying the application software's data from FLASH to the appropriate RAM locations. For this purpose the startup code typically uses functions from the compiler run-time libraries.

Trap Functions: Trap handling functions shall be activated when a microcontroller's hardware safety mechanism detects an anomaly, such as when the memory protection unit detects illegal memory access. The required behavior of trap handling functions is use-case specific, therefore the system integrator is responsible for the implementation and qualification of these functions. However, to ensure that when a hardware trap occurs, the correct trap processing function is called with the appropriate input parameters set to the correct values is part of the compiler tool qualification which is under responsibility of the tool supplier.

C Library Functions

Qualification Criteria: The C library contains functions that perform basic operations, such as memory operations, and low-level mathematical functions, such as square root, power functions, trigonometric functions, and so on. The functions of the C library are

described in the user documentation, are called from the user's application software and are integrated into the vehicle software. It is therefore “obvious” that the qualification requirements for software components apply. However, the devil is in the details, as shown below.
The C library's mathematical functions are typically highly optimized and therefore use functions from the runtime floating-point libraries. The runtime libraries were qualified in the context of a tool qualification, but now the intended use has changed – for example, the function could be called with input arguments that are outside the valid range which is used by the compiler – and software component qualification criteria apply to those runtime functions.
The C library functions declared in the header file "fenv.h" access and modify the so- called floating-point environment. The floating-point environment is part of the compiler runtime environment, and its configuration affects the behavior of compiler-generated code for floating point operations. For these functions, the question is therefore whether the qualification criteria for tools, or the qualification criteria for software components, or both should apply.

ASIL: The ASIL level for which specific C library functions are suited is debatable. Some C library features are inherently unsafe due to weaknesses in previous versions of the ISO C standard (C90) and the desire to keep the standard backwards compatible. To correct the mistakes of the past, the ISO C committee introduced Annex K, which includes safer variants of the unsafe functions. The MISRA C and CERT C Coding Guidelines provide guidelines for the safe and secure use of C library functions and recommend against using several parts of the library functions.
To determine for which ASIL a feature is suitable, the following reasoning can be used: if a function is developed in accordance with ISO 26262 ASIL-x requirements, then the function is qualified for use in ASIL-x software, independent of MISRA recommendations, provided the user adheres to the guidelines from the safety manual. For example, such a guideline could impose restrictions on the valid domain of a function's input parameters.

Content: The C library contains many functions that are unsafe and/or are not used in automotive software. Should a vendor include or remove such features from a qualified library? If such features are removed, the user's software may not build when switching from an unqualified library to a qualified library, and functionality such as printf()-style debugging or logging functions will no longer be available during the software development. Both issues can be considered as disadvantages. On the other hand, if a qualified library contains functions qualified for different ASILs, including QM, then some means must be provided to ensure that functions with an inappropriate level of qualification are not built into the production software.

Safety Documents - Practical Use - Costs

What has been said before about the “certified” versus “certifiable” compiler toolset qualification approaches also applies to software component / library qualification.

When the qualification of the compiler and its associated libraries has been conducted by the compiler supplier, the customer receives both a qualified compiler and qualified libraries, accompanied by associated safety manuals. The supplier can provide a third-party certificate as evidence that the qualification meets ISO 26262 requirements.
The formal evidence of the correct use of the compiler and integration of the library software should be created by the user via a coverage analysis showing that the guidelines/requirements from the safety manuals have been implemented.
In the ideal case the safety manuals are provided in a machine-readable format such as ReqIF. This enables the coverage analysis to be performed automatically by reading the guidelines into the company's requirements management system and addressing the guidelines through the company's standard operating processes. This prevents annoying and error-prone conversions and the adjustment to new work processes.
Ideally, the user should be able to reuse existing coverage analysis activities if the compiler/library is updated or if a switch is made to another type of microcontroller. Therefore, safety manuals should be prepared for comparison and merging. All forementioned items have been implemented in the TASKING Qualified C Library.

Conclusion

In conclusion, adhering to functional safety standards necessitates the qualification of both the compiler and its accompanying libraries. Qualifying these components demands an in-depth understanding of FuSa standards, the ISO C standard, and the compiler toolset's design and implementation. Therefore, the tool manufacturer is typically best equipped to undertake this qualification process.
For the end-user, the maturity and format of the provided safety documentation significantly impacts the cost-effectiveness of their engineering and FuSa compliance endeavors. Therefore, investing in well-documented and certified compiler toolsets and libraries can be a judicious decision. With many FuSa and software certification tasks handled by the compiler and library supplier, your efforts can be channeled towards advancing innovations in automotive software.

Author

Gerard Vink is Industry Specialist Product Definition at TASKING. He studied mechanical engineering and computer science and has long-term experience with the development of tools and processes for (embedded) software development.