Compiler Errors

Software compilers are complex software configuration items that translate source code into executable code, which becomes part of the safety-critical application. As such, any error in the compiler-generated executable code can introduce a failure in the safety-critical system. The goal of safety-risk mitigation plans is to detect and correct such errors. One could develop sufficient confidence in the compiler-generated object code by using a qualified compiler, or by using alternate mitigation activities to verify the object code.

Table 1 lists the major potential compiler errors expected to be mitigated using a qualified compiler. Included in the table are alternate mechanisms that would also mitigate for the same compiler output errors when not using a qualified compiler.

	Potential Compiler Errors in Object Code	Alternate Mitigation
1	Illegal sizes of memory allocated to variables of a given type	Verification test cases and robustness testing
2	Illegal branching instructions	Verification test cases
3	Load/store operation with incorrect source/destination addresses	Partitioning management, verification test cases, and robustness testing
4	Register move operations with incorrect source/ destination	Verification test cases
5	Unintended code	Source code to object code analysis
6	Incorrect branching	Verification test cases and MC/DC analysis
7	Incorrect code	Verification test cases
8	Unsatisfiable branches	Unreachable code
9	Missing expected insertion of code	Verification test cases

Table 1: Potential Compiler Errors and Alternate Mitigations

As shown in Table 1, when the generated code is verified in accordance with the safety guideline/standard (as an alternate to using a qualified compiler) there is a high degree of confidence that a malfunction and its corresponding erroneous output will be detected and prevented. This is achieved through performing the following activities on the software:

1. Requirements-based tests run on the executable object code, including robustness testing
2. Special focus on boundary values, etc.

3. Source code MC/DC coverage analysis
4. Source code to object code analysis (analyze compiler-added code such as complex libraries)

For ISO 26262 ASIL D, with these verification steps in place, a qualified compiler is not necessary. For other ASIL levels, one needs to choose between a qualified compiler (if available) or perform the activities above, even though some may be associated with a more stringent ASIL level.

Being qualified does not mean defect-free. For complex software like a compiler, you can never be 100% confident in defect detection, even if the tool is “qualified”. While a qualified compiler increases the probability of preventing errors in the output, it is merely a higher level of confidence, not an assurance that there would not be an erroneous output leading to a violation of a safety requirement.

A qualified ISO 26262:11 TCL-3 compiler may allow you to skip the source-to-object code analysis, but there are no formal certification credits in exchange for tool qualification offered by ISO 26262:11 or DO-330 compiler qualification.

The information contained in this document is for informational purposes only and is subject to change without notice. CoreAVI, the CoreAVI tracer logo, and combinations thereof are trademarks of CoreAVI. All other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.