VECTOR

Compiler Warning

Version 1.1.0
2018-07-31
Application Note AN-ISC-8-1184

Author Andreas Raisch, Markus Schwarz

Restrictions Customer Confidential – Vector decides

Abstract Warning free code is an important quality goal for embedded software. Nevertheless, compiler warnings can occur for highly configurable software. Typical compiler warnings are listed and justified within this document.

Table of Contents

1 Overview

MICROSAR BasicSoftware (BSW) is developed in a product line approach, independent from specific ECU projects or compilers. MICROSAR BSW supports the AUTOSAR MemoryAbstraction and CompilerAbstraction concepts and supports a huge range of different compiler vendors, versions and options.

MICROSAR BSW is delivered as static source code and code generators to support the ECU project specific adaption and optimization of the BSW behavior and feature set based on AUTOSAR configuration files and user defined selections.

Additionally, MICROSAR BSW supports compile time configuration, link time configuration and post-build configuration.

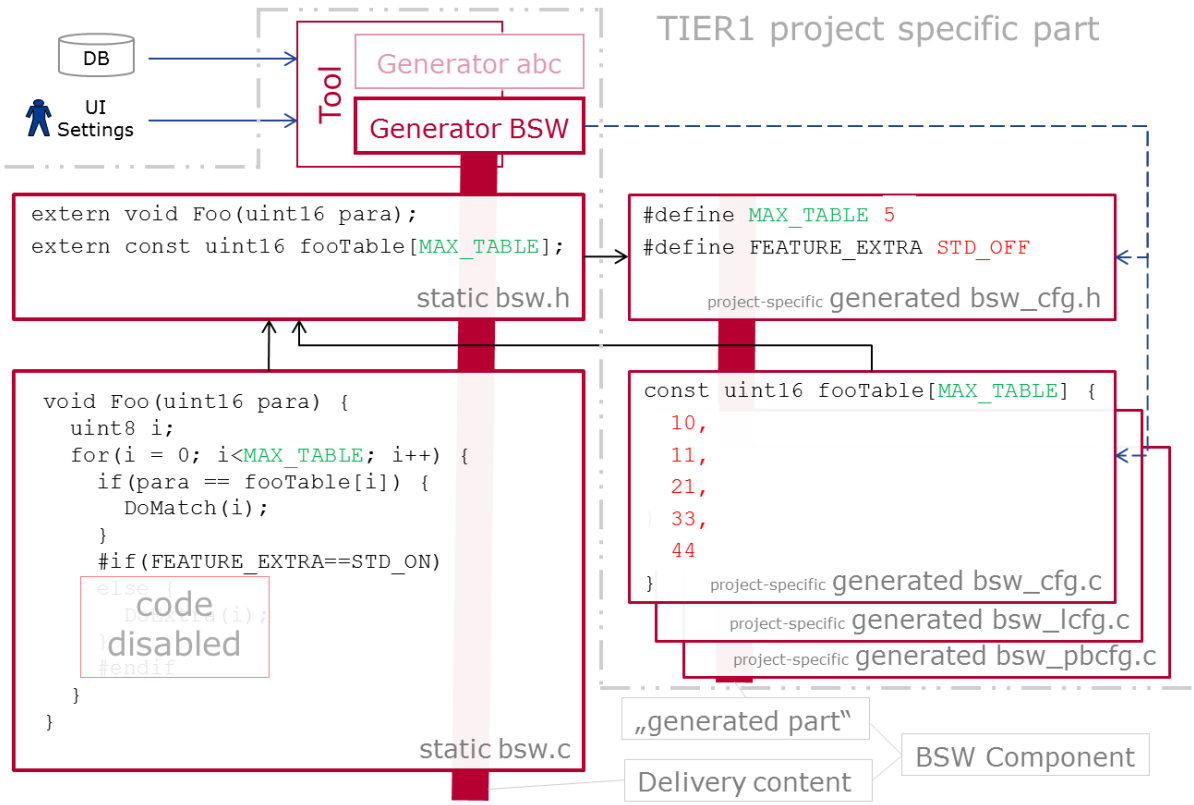

Figure 1 MICROSAR BSW Component

A general quality goal for MICROSAR BSW is “warning free code”. The coding style guide for MICROSAR BSW is based on MISRA-C:2004 and MISRA-C:2012. Checks for MISRA compliance are an essential part of our development process. (see [1], [3])

Nevertheless, we accept deviations to the “warning free code” goal.

There are some common deviations that are defined within this document. And there are accepted MISRA deviations (see [1], [3]) that might also be issued by some compilers depending on their warning level.

These deviations are caused by basic coding principles of our code. There is no ESCAN reporting for those deviations.

All other known and accepted deviations are reported within the delivery as ESCAN type “Warning in released product”.

Document [2] explains the relationship between user-selected code generation (optimization) options and compiler warnings.

Copyright © 2018 - Vector Informatik GmbH

Contact Information: www.vector.com or +49-711-80 670-0

1.1 Deviation Procedure

Priority	Rule	Rationale
1	MICROSAR BSW shall be compiler warning free. Thus, compiler warnings shall be prevented wherever possible.	Detection of compiler warning either during component development or more frequently when applying the customer specific compiler/options/BSW configuration during the delivery tests.
2	If a compiler warning is detected, the code shall be analyzed and, if reasonable, be changed/extended to become warning free.	We want to prevent to deliver defect code to our customers. We also need to take into account e.g. runtime efficiency, maintainability and standardized code for many use-cases.
3	If fixing the compiler warning is not a solution, an ESCAN of type “Warning in released product” shall be created to document the accepted deviation also in the report of known issues of deliveries. Exception: The deviations in this document do not require an ESCAN, as they are reported to the customer with this document.	See list of accepted deviations in this document.

Table 1 Priority of measures to prevent compiler warnings

1.2 Default BSW Delivery Process

The configuration and compilation and linkage during the delivery test activities yield most of the compiler warnings, because the development tool chain and settings specified by the customer are used.

The process in short is:

- Customer specifies the compiler brand, version and options (via Questionnaire) and provides more information on the expected use-case
- Delivery engineer creates example configurations of the BSW and compiles and links the outcome
- Compiler warnings are analyzed and either documented as ESCAN of type “Warning in released product” or reported to the development teams to trigger a fix.
- ESCANs of type “Issue in released product” and “Warning in released product” are reported within the delivery documentation as “known issues”

2 Accepted Deviations

Deviations in this chapter are typically accepted as “Warning in released product” when

- it has been checked that no incorrect behavior at ECU runtime occurs
- no simple remedy exists

Note
If a compiler warning has passed the analyzing steps according Table 1 Priority of measures to prevent compiler warnings and the result is still “accepted deviation”, the compiler warning will not be fixed in the product.

Copyright © 2018 - Vector Informatik GmbH

Contact Information: www.vector.com or +49-711-80 670-0

2.1 Unused/Unreferenced Parameter/Argument

Deviation ID	CW_001
Example compiler warning strings	„Unreferenced formal parameters“, „parameter is never used“, „has no-used argument“
Reason	BSW provides APIs as described by standards. Not all parameters are necessary and used within the function in all possible usage scenarios.
Configuration dependent	Yes
Potential risk	The function contains unused code.
Prevention of risk	Code inspection has to check that the unused code is not critical concerning the expected behavior. Small increase of ROM footprint is accepted.
Note	Workarounds for suppressing such compiler warnings often conflict with MISRA-C:2004, rule 14.2 (See MISRA Compliance Documentation, Deviation ID MD_MSR_14.2)

Table 2 CW_001

2.2 Unused/Unreferenced Define, Enum Value

Deviation ID	CW_002
Example compiler warning strings	„Unused enumeration values“, „Unused define value“
Reason	Encapsulation of defines, enum-values and similar items for multiple complex configuration data sets (via #ifdef) reduces readability and maintainability.
Configuration dependent	Yes
Potential risk	Defined but never used enums or defines may decrease readability and maintainability.
Prevention of risk	Execution of runtime tests with different configuration variants.
Note	-

Table 3 CW_002

Copyright © 2018 - Vector Informatik GmbH

Contact Information: www.vector.com or +49-711-80 670-0

2.3 Unused/Unreferenced Variable

Deviation ID	CW_003
Example compiler warning strings	„unreferenced local variable within abc.c“, „variable "abc" was declared but never referenced“, „Variables related to message supervision are set but never used“,
Reason	Because of goal "minimize RAM footprint", unused variables shall be disabled via #ifdef based on configuration data sets. Nevertheless, warning can occur for rare cases where encapsulation for multiple complex configuration data sets (via #ifdef) reduces readability and maintainability.
Configuration dependent	Yes
Potential risk	The function contains unused code.
Prevention of risk	Code inspection has to check that the unused variable is not critical concerning the expected behavior. Small increase of RAM footprint is accepted.
Note	-

Table 4 CW_003

2.4 Unused/Unreferenced Function

Deviation ID	CW_004
Example compiler warning strings	„Declared but unused function abc“, „unused static function abc“
Reason	BSW provides APIs as described by standard. Because of goal "minimize ROM footprint", unused functions shall be disabled via #ifdef based on configuration data sets. Not all provided functions might be used in the concrete ECU project (configuration and usage dependent). Encapsulation for multiple complex configuration data sets (via #ifdef) reduces readability and maintainability and is thus not provided on a per-function base for all APIs.
Configuration dependent	Yes
Potential risk	The project contains unused code.
Prevention of risk	See “Reason”
Note	Partly addressed by MISRA-C:2004, rule 8.10 (See MISRA Compliance Documentation, Deviation ID MD_MSR_8.10)

Table 5 CW_004

Copyright © 2018 - Vector Informatik GmbH

Contact Information: www.vector.com or +49-711-80 670-0

2.5 Condition Evaluates Always to True/False

Deviation ID	CW_005
Example compiler warning strings	„conditional expression is constant“, „conditional expression or part of it is always true/false / statement not reached“, „condition is always true/false“, „compare out of range / condition is always true/false“, „The result of this logical operation is always 'false' or ‘true’“
Reason	Typically caused by an if-statement applied on external configuration data. Configuration data is const for the given compilation context but might be changed at link-time or post-build time.
Configuration dependent	Yes
Potential risk	The function contains useless conditions with possibly dead code.
Prevention of risk	The code inspection is in charge to distinguish between useless conditions with possibly dead code and correct code as described in “reason”.
Note	-

Table 6 CW_005

2.6 Unreachable Code/Statement

Deviation ID	CW_006
Example compiler warning strings	„conditional expression or part of it is always true/false / statement not reached“, „statement will never be executed“, „this switch default label is unreachable“
Reason	Because of goal "minimize ROM footprint", unreachable code shall be disabled via #ifdef based on configuration data sets. Nevertheless, warning might occur as secondary effect on e.g. "condition evaluates to true/false"
Configuration dependent	Yes
Potential risk	The function contains unused code.
Prevention of risk	Code inspection has to check that the unused code is not critical concerning the expected behavior. Small increase of ROM footprint is accepted.
Note	Addressed by MISRA-C:2004, rule 14.1 (See MISRA Compliance Documentation, Deviation ID MD_MSR_14.1)

Table 7 CW_006

Copyright © 2018 - Vector Informatik GmbH

Contact Information: www.vector.com or +49-711-80 670-0

2.7 Dead Assignment / Variable Set but Not Used

Deviation ID	CW_007
Example compiler warning strings	“Variable 'abc' was set but never used“, „"local variable is initialized but not referenced", „dead assignment to "abc" eliminated“, „Dead assignment (subtraction with zero)“, „Removed dead assignment“ “Useless assignment to variable …. Assigned value not used.”
Reason	Variable shall be always assigned a valid value. Overwriting the value before use can therefore happen based on the assumed execution path. This can especially occur for the DUMMY_STATEMENT macro we use or for the return value assignment in our API pattern.
Configuration dependent	Yes
Potential risk	The function contains unused code and variables.
Prevention of risk	Code inspection has to check that the unused code is not critical concerning the expected behavior. Small increase of ROM/RAM footprint is accepted.
Note	-

Table 8 CW_007

2.8 ASM Statements Used

Deviation ID	CW_008
Example compiler warning strings	“asm statements used“
Reason	MICROSAR is written in C but might use compiler specific ASM statements in rare cases. Usage of ASM is strictly limited by coding style guide.
Configuration dependent	No
Potential risk	ASM syntax is compiler and compiler version dependent.
Prevention of risk	Component release test and delivery test shows correct behavior.
Note	-

Table 9 CW_008

3 Additional Resources

[1] Vectors AN-ISC-8-1213 Compliance Documentation MISRA-C:2004, Version 3.0.0 or later

[2] Vectors Technical Reference MICROSAR ComStackLib, Version 2.0.0 or later

[3] Vectors AN-ISC-8-1225 MISRA-C Compliance, Version 1.0.0 or later

4 Contacts

For a full list with all Vector locations and addresses worldwide, please visit http://vector.com/contact/.

Copyright © 2018 - Vector Informatik GmbH

Contact Information: www.vector.com or +49-711-80 670-0