CodeWarrior Development Studio for S12(X) V5.x

MISRA-C:2004 Compliance Exceptions for the S12(X) and XGATE Libraries

Revised: November 03, 2015

How to Contact Us

Corporate Headquarters	Freescale Semiconductor, Inc. 6501 William Cannon Drive West Austin, TX 78735 U.S.A.
World Wide Web	`http://www.freescale.com/codewarrior`
Technical Support	`http://www.freescale.com/support`

Table of Contents

1 Introduction

The CodeWarrior Development Studio for S12(x) V5.x MISRA-C:2004 Compliance Exceptions for the S12(X) and XGATE Libraries manual covers the MISRA-C:2004 compliance exceptions for the HCS12 and XGATE libraries.

This document contains following chapters:

Chapter 2 - HCS12 contains the list of MISRA-C:2004 exceptions for HCS12

Chapter 3 - XGATE contains the lists of MISRA-C:2004 exceptions for XGATE

Appendix A - References contains the list of targets for HCS12 and XGATE

For a particular target, either HCS12 or XGATE, the exceptions to MISRA rules are grouped into general exceptions, which apply across all the library projects, and per-project exceptions, which are the exceptions associated with a certain library project.

2

HCS12

This chapter contains these topics for HCS12:

- Inline Assembly
- General Exceptions
- Per-project Exceptions

Inline Assembly

Inline assembly is altogether ignored when checking for MISRA-C:2004 compliancy.

General Exceptions

Table 1. lists the exceptions to MISRA-C:2004 rules that apply across all the library projects.

Table 1. HCS12 general library exceptions to MISRA-C:2004 rules

MISRA-C:2004 Rule	Exception
6.3 ADV	inhibit the message on the use of a modifier or a type outside of a typedef
19.7 ADV	allow function-like macros
19.15 REQ	allow repeatedly included header files - all the libray headers are guarded using macros
14.7 REQ	allow multiple exit points for functions
14.5 REQ	allow 'continue' statements

Table 1. HCS12 general library exceptions to MISRA-C:2004 rules

MISRA-C:2004 Rule	Exception
18.4 REQ	allow unions
20.4 REQ, 20.5 REQ	accept several deprecated symbols

Per-project Exceptions

Elective Note #960, MISRA 19.6 REQ: #undef required to support non-ANSI pointer qualifiers 'near' and 'far'

`default2.sgm: 2 [1]`

Warning #537: allow multiple use

`default2.sgm: 27 [1]`

Elective Note #961, MISRA 19.3 ADV: '#' is used in HLI as an operator - see pragma NO_STRING_CONSTR above

`hidef.h: 41 [1]`

Elective Note #961, MISRA 19.3 ADV: '#' is used in HLI as an operator - see pragma NO_STRING_CONSTR above

`hidef.h: 117 [1]`

Warning #683: inhibit warning on standard function being #define'd

`stdlib.h: 45 [2]`

Elective Note #960, MISRA 16.3 REQ: message reported not for a function, but for a function pointer

`stdlib.h: 82 [2]`

Elective Note #960, MISRA 16.3 REQ: message reported not for a function, but for a function pointer

`stdlib.h: 85 [2]`

Informational #715: name not used

`assert.c: 21 [2]`

Informational #766: hidef.h contains conditionally compiled code

`assert.c: 24 [2]`

Elective Note #960, MISRA 10.1 REQ: the conversion has no impact on bit pattern intepretation

`ctype.c: 11 [2]`

Elective Note #960, MISRA 19.6 REQ: character classification macros must be undefined prior to defining the corresponding character classification functions

`ctype.c: 276 [2]`

Informational #766: hidef.h contains conditionally compiled code

`ctype.c: 368 [2]`

Informational #773: va_end is never used as an expression operand

`stdarg.h: 120 [2]`

Warning #683: inhibit warning on standard function being #define'd

`stdio.h: 76 [2]`

Elective Note #960, MISRA 16.1 REQ: standard library function implementation

`stdio.h: 149 [2]`

Elective Note #960, MISRA 16.1 REQ: standard library function implementation

`stdio.h: 150 [2]`

Elective Note #960, MISRA 16.1 REQ: standard library function implementation

`stdio.h: 157 [2]`

Elective Note #960, MISRA 16.1 REQ: standard library function implementation

`stdio.h: 158 [2]`

Elective Note #960, MISRA 16.1 REQ: standard library function implementation

`stdio.h: 160 [2]`

Elective Note #960, MISRA 16.3 REQ: message reported for a function pointer parameter

`stdio.h: 163 [2]`

Elective Note #960, MISRA 16.1 REQ: standard library function implementation

`stdio.h: 204 [2]`

Elective Note #960, MISRA 16.1 REQ: standard library function implementation

`embedded.c: 31 [2]`

Warning #643: misleading warning ('&format' does not have a far type)

`embedded.c: 36 [2]`

Elective Note #928, MISRA 11.4 ADV: safe conversion to 'char *'

`embedded.c: 37 [2]`

Warning #438: 'va_end' must be invoked before return in a variadic function

`embedded.c: 41 [2]`

Elective Note #960, MISRA 16.1 REQ: standard library function implementation

`embedded.c: 44 [2]`

Warning #643: misleading warning ('&format' does not have a far type)

`embedded.c: 49 [2]`

Elective Note #928, MISRA 11.4 ADV: safe conversion to 'char *'

`embedded.c: 50 [2]`

Warning #438: 'va_end' must be invoked before return in a variadic function

`embedded.c: 55 [2]`

Warning #625: accept unusual type modifier

`embedded.c: 64 [2]`

Elective Note #923, MISRA 11.3 ADV: no support for multiple file descriptors

`embedded.c: 91 [2]`

Informational #715: this is the implementation of a standard library function, so its prototype is left unchanged

`embedded.c: 92 [2]`

Elective Note #926, MISRA 11.4 ADV: the conversion is safe

`embedded.c: 95 [2]`

Informational #715, MISRA 16.7 ADV: standard library function implementation

`embedded.c: 97 [2]`

Informational #818: standard library function implementation

`embedded.c: 97 [2]`

Elective Note #960, MISRA 16.1 REQ: standard library function implementation

`embedded.c: 103 [2]`

Warning #643: misleading warning ('&format' does not have a far pointer)

`embedded.c: 108 [2]`

Elective Note #928, MISRA 11.4 ADV: safe conversion to 'char *'

`embedded.c: 109 [2]`

Warning #438: 'va_end' must be invoked before return in a variadic function

`embedded.c: 113 [2]`

Informational #715, MISRA 16.7 ADV: standard library function implementation

`embedded.c: 114 [2]`

Informational #818: standard library function implementation

`embedded.c: 114 [2]`

Informational #715, MISRA 16.7 ADV: standard library function implementation

`embedded.c: 119 [2]`

Informational #818: standard library function implementation

`embedded.c: 119 [2]`

Informational #715, MISRA 16.7 ADV: standard library function implementation

`embedded.c: 124 [2]`

Informational #818: standard library function implementation

`embedded.c: 124 [2]`

Informational #715, MISRA 16.7 ADV: standard library function implementation

`embedded.c: 128 [2]`

Informational #818: standard library function implementation

`embedded.c: 128 [2]`

Informational #715: this is the implementation of a standard library function, so its prototype is left unchanged

`embedded.c: 132 [2]`

Warning #511: pointer size depends on both the target and the memory model

`embedded.c: 152 [2]`

Elective Note #923, MISRA 11.3 ADV: safe conversion

`embedded.c: 153 [2]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: the boolean operation results from the expansion of macro 'CONVERT_TO_PAGED'

`embedded.c: 154 [2]`

Elective Note #960, MISRA 12.7 REQ: the signed quantity is always positive

`embedded.c: 155 [2]`

Elective Note #960, MISRA 14.3 REQ: the macro expands to HLI several statements

`embedded.c: 165 [2]`

Elective Note #960, MISRA 14.3 REQ: the macro expands to HLI several statements

`embedded.c: 169 [2]`

Informational #715: this is the implementation of a standard library function, so its prototype is left unchanged

`embedded.c: 170 [2]`

Informational #766: hidef.h contains conditionally compiled code

`embedded.c: 172 [2]`

Elective Note #960, MISRA 10.1 REQ: the result of sizeof() has type size_t

`heap.c: 20 [2]`

Warning #414: division by zero!

`heap.c: 32 [2]`

Warning #564: division by zero!

`heap.c: 32 [2]`

Elective Note #931: division by zero!

heap.c: 32 [2]

Informational #818, MISRA 16.7 ADV: symbol element not referenced

heap.c: 33 [2]

Informational #715: symbol element not referenced

heap.c: 33 [2]

Warning #438: symbol element not referenced

heap.c: 33 [2]

Informational #715: inhibit message on 'a' and 'p' not being referenced

locale.c: 17 [2]

Elective Note #960, MISRA 19.6 REQ: #undef required to support non-ANSI pointer qualifiers 'near' and 'far'

non_bank.sgm: 1 [1]

Warning #537: allow multiple use

non_bank.sgm: 24 [1]

Elective Note #960, MISRA 19.6 REQ: #undef required to support non-ANSI pointer qualifiers 'near' and 'far'

default.sgm: 1 [1]

Warning #537: allow multiple use

default.sgm: 24 [1]

Informational #708: initialization applied to the first named member of the union

math.c: 75 [3]

Informational #708: initialization applied to the first named member of the union

math.c: 77 [3]

Informational #777: the purpose of the test is to compare the bit patterns for an exact match

`math.c: 185 [4]`

Elective Note #934: no dynamic linking, an absolute address is obtained

`math.c: 494 [4]`

Elective Note #960, MISRA 12.4 REQ: no impact if 'fabs' is not called during expression evaluation

`math.c: 495 [4]`

Informational #750: suppress the messages on several local macros not being referenced

`math.c: 804 [4]`

Informational #777: the purpose of the test is to compare the bit patterns for an exact match

`math.c: 1205 [4]`

Informational #777: the purpose of the test is to compare the bit patterns for an exact match

`math.c: 1215 [4]`

Informational #766: hidef.h contains conditionally compiled code

`math.c: 1234 [4]`

Informational #766: hidef.h contains conditionally compiled code

`mathf.c: 1145 [4]`

Elective Note #960, MISRA 16.3 REQ: this is a function pointer declaration

`printf.c: 83 [2]`

Elective Note #960, MISRA 7.1 REQ: safe use of octal constants

`printf.c: 103 [2]`

Warning #625: (accept unusual type modifier; no precision loss)

`printf.c: 123 [2]`

Warning #619: (accept unusual type modifier; no precision loss)

`printf.c: 123 [2]`

Warning #619: no precision loss

`printf.c: 128 [2]`

Informational #702, MISRA 12.7 REQ: allow signed right shift, its positive anyway

`printf.c: 137 [2]`

Elective Note #960: allow signed right shift, its positive anyway

`printf.c: 137 [2]`

Informational #702, MISRA 12.7 REQ: allow signed right shift, its positive anyway

`printf.c: 150 [2]`

Elective Note #960: allow signed right shift, its positive anyway

`printf.c: 150 [2]`

Informational #750: suppress the message on macros 'DIGITS' and 'BOUND' not being referenced

`printf.c: 167 [2]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`printf.c: 275 [2]`

Elective Note #960, MISRA 10.5 REQ: '~' applied to an operand of type 'unsigned int'

`printf.c: 322 [2]`

Elective Note #960, MISRA 10.5 REQ: '~' applied to an operand of type 'unsigned int'

`printf.c: 325 [2]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`printf.c: 330 [2]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`printf.c: 346 [2]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`printf.c: 400 [2]`

Informational #826: safe conversion

`printf.c: 401 [2]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`printf.c: 405 [2]`

Informational #826: safe conversion

`printf.c: 406 [2]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`printf.c: 410 [2]`

Informational #826: safe conversion

`printf.c: 411 [2]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`printf.c: 419 [2]`

Elective Note #926, MISRA 11.4 ADV: conversion is necessary and safe

printf.c: 426 [2]

Informational #801: Use of goto is not deprecated

printf.c: 428 [2]

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

printf.c: 447 [5]

Warning #613: Possible use of null pointer 'str' in argument

printf.c: 478 [2]

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: the Boolean value is target-dependent

printf.c: 493 [2]

Informational #774: the Boolean value is target-dependent

printf.c: 493 [2]

Informational #801: Use of goto is not deprecated

printf.c: 505 [2]

Elective Note #960, MISRA 10.5 REQ: '~' applied to an operand of type 'unsigned int'

printf.c: 516 [2]

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

printf.c: 523 [2]

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

printf.c: 528 [2]

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: target-dependent Boolean expressions

printf.c: 533 [2]

Informational #774: target-dependent Boolean expressions

`printf.c: 533 [2]`

Informational #845: target-dependent Boolean expressions

`printf.c: 533 [2]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`printf.c: 535 [2]`

Elective Note #923, MISRA 11.3 ADV: the cast is necessary, see comment above

`printf.c: 536 [2]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`printf.c: 539 [2]`

Informational #801: Use of goto is not deprecated

`printf.c: 547 [2]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`printf.c: 553 [2]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`printf.c: 558 [2]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`printf.c: 563 [2]`

Elective Note #960: the result of sizeof() is size_t, which is defined to 'unsigned int'

`printf.c: 579 [2]`

Elective Note #960: the result of sizeof() is size_t, which is defined to 'unsigned int'

`printf.c: 581 [2]`

Elective Note #960: the result of sizeof() is size_t, which is defined to 'unsigned int'

`printf.c: 583 [2]`

Warning #661: (no out-of-bounds access)

`printf.c: 714 [2]`

Warning #662: (no out-of-bounds access)

`printf.c: 714 [2]`

Warning #661: (no out-of-bounds access)

`printf.c: 716 [2]`

Informational #825: fallthrough is deliberate

`printf.c: 764 [2]`

Informational #801: Use of goto is not deprecated

`printf.c: 766 [2]`

Informational #801: Use of goto is not deprecated

`printf.c: 770 [2]`

Informational #825: fall through is deliberate

`printf.c: 775 [2]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`printf.c: 782 [2]`

Elective Note #960: the type of the result of sizeof() is size_t, which is defined to 'unsigned int'

`printf.c: 815 [2]`

Elective Note #960: the type of the result of sizeof() is size_t, which is defined to 'unsigned int'

`printf.c: 825 [2]`

Elective Note #960: the type of the result of sizeof() is size_t, which is defined to 'unsigned int'

`printf.c: 826 [2]`

Elective Note #960, MISRA 10.5 REQ: '~' applied to an operand of type 'unsigned int'

`printf.c: 842 [2]`

Elective Note #960: the type of the result of sizeof() is size_t, which is defined to 'unsigned int'

`printf.c: 874 [2]`

Elective Note #960: the type of the result of sizeof() is size_t, which is defined to 'unsigned int'

`printf.c: 875 [2]`

Elective Note #960, MISRA 10.5 REQ: '~' applied to operand of type 'unsigned int')

`printf.c: 885 [2]`

Elective Note #960: the type of the result of sizeof() is size_t, which is defined to 'unsigned int'

`printf.c: 888 [2]`

Elective Note #960: the type of the result of sizeof() is size_t, which is defined to 'unsigned int'

`printf.c: 898 [2]`

Elective Note #960: the type of the result of sizeof() is size_t, which is defined to 'unsigned int'

`printf.c: 900 [2]`

Elective Note #960: the type of the result of sizeof() is size_t, which is defined to 'unsigned int'

`printf.c: 919 [2]`

Elective Note #960: the type of the result of sizeof() is size_t, which is defined to 'unsigned int'

`printf.c: 991 [2]`

Elective Note #960: the type of the result of sizeof() is size_t, which is defined to 'unsigned int'

`printf.c: 992 [2]`

Informational #825: fallthrough is deliberate

`printf.c: 1066 [2]`

Informational #818, MISRA 16.7 ADV: this is a standard library function, cannot change its prototype

`printf.c: 1073 [2]`

Elective Note #960, MISRA 16.3 REQ: this is a function pointer parameter

`printf.c: 1149 [2]`

Elective Note #960, MISRA 16.1 REQ: standard library function implementation

`printf.c: 1190 [2]`

Warning #643: misleading warning ('&format' does not have a far pointer type)

`printf.c: 1196 [2]`

Elective Note #928, MISRA 11.4 ADV: safe conversion to 'char *'

`printf.c: 1197 [2]`

Warning #438: 'va_end' must be invoked before return in a variadic function

`printf.c: 1202 [2]`

Informational #766: header file contains conditionally compiled code

`printf.c: 1215 [2]`

Warning #625: accept unusual type modifier

`scanf.c: 104 [2]`

Warning #625: accept unusual type modifier

`scanf.c: 154 [2]`

Warning #625: accept unusual type modifier

`scanf.c: 157 [2]`

Warning #625: accept unusual type modifier

`scanf.c: 161 [2]`

Informational #801: Use of goto is not deprecated

`scanf.c: 184 [2]`

Elective Note #931: the expression is safe

`scanf.c: 191 [2]`

Informational #801: Use of goto is not deprecated

`scanf.c: 195 [2]`

Informational #801: Use of goto is not deprecated

`scanf.c: 199 [2]`

Informational #801: Use of goto is not deprecated

`scanf.c: 228 [2]`

Informational #801: Use of goto is not deprecated

`scanf.c: 237 [2]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`scanf.c: 262 [2]`

Informational #826: safe conversion

`scanf.c: 263 [2]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`scanf.c: 268 [2]`

Informational #826: safe conversion

`scanf.c: 269 [2]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`scanf.c: 273 [2]`

Informational #826: safe conversion

`scanf.c: 274 [2]`

Informational #801: Use of goto is not deprecated

`scanf.c: 286 [2]`

Informational #801: Use of goto is not deprecated

`scanf.c: 294 [2]`

Informational #801: Use of goto is not deprecated

`scanf.c: 305 [2]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`scanf.c: 338 [2]`

Informational #826: safe conversion

`scanf.c: 339 [2]`

Elective Note #961, MISRA 17.5 ADV: multiple indirection levels necessary in order to implement support for '%p'

`scanf.c: 343 [2]`

Warning #511: pointer size depends on both the target and the memory model (if truncation occurs, it is expected)

`scanf.c: 344 [2]`

Elective Note #923: pointer size depends on both the target and the memory model (if truncation occurs, it is expected)

`scanf.c: 344 [2]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`scanf.c: 345 [2]`

Informational #826: safe conversion

`scanf.c: 346 [2]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`scanf.c: 360 [2]`

Informational #826: safe conversion

`scanf.c: 361 [2]`

Informational #801: Use of goto is not deprecated

`scanf.c: 379 [2]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`scanf.c: 398 [6]`

Informational #826: safe conversion

`scanf.c: 399 [6]`

Warning #539: conditionally compiled 'if' clause

`scanf.c: 413 [2]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`scanf.c: 436 [2]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`scanf.c: 481 [2]`

Elective Note #946, MISRA 17.2 REQ, MISRA 17.3 REQ: the two pointers point into the same array object

`scanf.c: 521 [2]`

Elective Note #947: the two pointers point into the same array object

`scanf.c: 521 [2]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`scanf.c: 525 [2]`

Informational #826: safe conversion

`scanf.c: 526 [2]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`scanf.c: 530 [2]`

Informational #826: safe conversion

`scanf.c: 531 [2]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`scanf.c: 535 [2]`

Informational #826: safe conversion

`scanf.c: 536 [2]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`scanf.c: 555 [2]`

Elective Note #929, MISRA 11.4 ADV: conversion is safe

`scanf.c: 569 [2]`

Warning #539: conditionally compiled 'if' clause

`scanf.c: 570 [2]`

Informational #818, MISRA 16.7 ADV: standard library function, cannot change its prototype

`scanf.c: 592 [2]`

Elective Note #960, MISRA 16.1 REQ: standard library function implementation

`scanf.c: 597 [2]`

Warning #643: misleading warning ('&format' does not have a far pointer type)

`scanf.c: 601 [2]`

Elective Note #928, MISRA 11.4 ADV: safe conversion to 'char *'

`scanf.c: 602 [2]`

Warning #438: 'va_end' must be invoked before return in a variadic function

`scanf.c: 606 [2]`

Elective Note #961, MISRA 19.13 ADV: '#' is used in HLI as an operator - see pragma NO_STRING_CONSTR above

`signal.c: 17 [2]`

Elective Note #960, MISRA 14.3 REQ: macro HALTX expands to several HLI statements

`signal.c: 28 [2]`

Elective Note #923, MISRA 11.3 ADV: safe cast (no truncation) because the maximum signal number is 23 (SIGALRM)

`signal.c: 38 [2]`

Elective Note #923, MISRA 11.3 ADV: safe casts

`signal.c: 39 [2]`

Elective Note #929, MISRA 11.4 ADV: safe casts

`signal.c: 39 [2]`

Elective Note #923, MISRA 11.3 ADV: safe casts

`signal.c: 45 [2]`

Elective Note #929, MISRA 11.4 ADV: safe casts

`signal.c: 45 [2]`

Elective Note #960, MISRA 14.3 REQ: macro HALTX expands to several HLI statements

`signal.c: 54 [2]`

Informational #715: the function contains inline assembly only

`signal.c: 56 [2]`

Informational #766: non_bank.sgm is not a regular header file, it contains a conditionally CODE_SEG pragma

`signal.c: 61 [2]`

Elective Note #961, MISRA 19.13 ADV: '#' used not as the stringification preprocessing operator, but as an inline assembly

`operator`

`stdlib.c: 25 [2]`

Elective Note #960, MISRA 14.3 REQ: the macro expands to several HLI statements

`stdlib.c: 73 [2]`

Informational #715: standard library function implementation

`stdlib.c: 75 [2]`

Elective Note #960, MISRA 14.3 REQ: the macro expands to several HLI statements

`stdlib.c: 79 [2]`

Informational #715: standard library function implementation

`stdlib.c: 81 [2]`

Elective Note #960, MISRA 14.3 REQ: the macro expands to several HLI statements

`stdlib.c: 85 [2]`

Informational #715: standard library function implementation

`stdlib.c: 87 [2]`

Elective Note #960, MISRA 14.3 REQ: the macro expands to several HLI statements

`stdlib.c: 91 [2]`

Informational #715: standard library function implementation

`stdlib.c: 93 [2]`

Elective Note #960, MISRA 14.3 REQ: the macro expands to several HLI statements

stdlib.c: 97 [2]

Informational #715, MISRA 16.7 ADV: standard library function implementation

stdlib.c: 99 [2]

Informational #818: standard library function implementation

stdlib.c: 99 [2]

Elective Note #960, MISRA 14.3 REQ: the macro expands to several HLI statements

stdlib.c: 103 [2]

Informational #715, MISRA 16.7 ADV: standard library function implementation

stdlib.c: 105 [2]

Informational #818: standard library function implementation

stdlib.c: 105 [2]

Elective Note #960, MISRA 14.3 REQ: the macro expands to several HLI statements

stdlib.c: 109 [2]

Informational #715, MISRA 16.7 ADV: standard library function implementation

stdlib.c: 111 [2]

Informational #818: standard library function implementation

stdlib.c: 111 [2]

Elective Note #960, MISRA 14.3 REQ: the macro expands to several HLI statements

stdlib.c: 115 [2]

Informational #715, MISRA 16.7 ADV: standard library function implementation

`stdlib.c: 117 [2]`

Informational #818: standard library function implementation

`stdlib.c: 117 [2]`

Warning #625: accept unusual type modifier

`stdlib.c: 145 [2]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 156 [2]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 162 [2]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 171 [2]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 175 [2]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 186 [2]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 190 [2]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 207 [2]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 211 [2]`

Elective Note #960, MISRA 14.3 REQ: the macro does not always expand to a null statement

`stdlib.c: 215 [2]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 219 [2]`

Elective Note #926, MISRA 11.4 ADV: safe cast

`stdlib.c: 225 [2]`

Warning #625: accept unusual type modifier

`stdlib.c: 246 [2]`

Warning #610: pointer tested against NULL

`stdlib.c: 252 [2]`

Elective Note #926, MISRA 11.4 ADV: safe cast

`stdlib.c: 253 [2]`

Warning #610: pointer tested against NULL

`stdlib.c: 263 [2]`

Elective Note #926, MISRA 11.4 ADV: safe cast

`stdlib.c: 264 [2]`

Elective Note #946, MISRA 17.2 REQ, MISRA 17.3 REQ: the two pointers point into the same array object

`stdlib.c: 272 [2]`

Elective Note #947: the two pointers point into the same array object

`stdlib.c: 272 [2]`

Warning #610: pointer tested against NULL

`stdlib.c: 278 [2]`

Elective Note #926, MISRA 11.4 ADV: safe cast

`stdlib.c: 279 [2]`

Warning #625: accept unusual type modifier

`stdlib.c: 302 [2]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 319 [2]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 326 [2]`

Elective Note #960, MISRA 14.3 REQ: the macro does not always expand to a null statement

`stdlib.c: 330 [2]`

Elective Note #960, MISRA 14.3 REQ: the macro does not always expand to a null statement

`stdlib.c: 332 [2]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 338 [2]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 341 [2]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 345 [2]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 347 [2]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 350 [2]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 365 [2]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 389 [2]`

Elective Note #960, MISRA 14.3 REQ: the macro does not always expand to a null statement

`stdlib.c: 395 [2]`

Elective Note #960, MISRA 14.3 REQ: the macro does not always expand to a null statement

`stdlib.c: 403 [2]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 433 [2]`

Elective Note #960, MISRA 14.3 REQ: the macro does not always expand to a null statement

`stdlib.c: 440 [2]`

Elective Note #960, MISRA 14.3 REQ: the macro does not always expand to a null statement

`stdlib.c: 442 [2]`

Elective Note #960, MISRA 14.3 REQ: the macro does not always expand to a null statement

`stdlib.c: 449 [2]`

Elective Note #960, MISRA 14.3 REQ: the macro does not always expand to a null statement

`stdlib.c: 451 [2]`

Elective Note #960, MISRA 14.3 REQ: the macro does not always expand to a null statement

`stdlib.c: 458 [2]`

Elective Note #960, MISRA 14.3 REQ: the macro does not always expand to a null statement

`stdlib.c: 460 [2]`

Elective Note #926, MISRA 11.4 ADV: safe cast

`stdlib.c: 477 [2]`

Warning #625: accept unusual type modifier

`stdlib.c: 539 [2]`

Elective Note #960, MISRA 16.3 REQ: function pointer parameter

`stdlib.c: 610 [2]`

Warning #625: accept unusual type modifier

`stdlib.c: 614 [2]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`stdlib.c: 645 [2]`

Elective Note #927, MISRA 11.4 ADV: deliberate cast from 'char*' to a word pointer type

`stdlib.c: 648 [2]`

Informational #826: deliberate cast from 'char *' to a word pointer type

`stdlib.c: 649 [2]`

Elective Note #960, MISRA 16.3 REQ: function pointer parameter

`stdlib.c: 667 [2]`

Warning #625: accept unusual type modifier

`stdlib.c: 670 [2]`

Elective Note #931: safe expression

`stdlib.c: 678 [2]`

Elective Note #960, MISRA 12.4 REQ: safe expression

`stdlib.c: 680 [2]`

Informational #766: header files 'math.h' and 'float.h' contain conditionally compiled code

`stdlib.c: 778 [2]`

Elective Note #926, MISRA 11.4 ADV: deliberate cast

`string.c: 17 [2]`

Informational #715: implementation of a standard library function

`string.c: 19 [2]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 22 [2]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 34 [2]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 35 [2]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 58 [2]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 59 [2]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 76 [2]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 77 [2]`

Elective Note #946, MISRA 17.2 REQ, MISRA 17.3 REQ: memmove implementation : need to establish if the two memory areas

`overlap`

`string.c: 79 [2]`

Elective Note #946, MISRA 17.2 REQ, MISRA 17.3 REQ: memmove implementation : need to establish if the two memory areas

`overlap`

`string.c: 83 [2]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 95 [2]`

Elective Note #926, MISRA 11.4 ADV: safe conversion to 'char *'

`string.c: 118 [2]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 119 [2]`

Elective Note #946, MISRA 17.2 REQ, MISRA 17.3 REQ: the two pointers point into the same string

`string.c: 124 [2]`

Elective Note #947, MISRA 17.2 REQ: the two pointers point into the same string

`string.c: 125 [2]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 131 [2]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 140 [2]`

Informational #720, MISRA 13.1 REQ, MISRA 13.2 ADV: assignment deliberately used in a Boolean context

`string.c: 143 [2]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 148 [2]`

Informational #820, MISRA 13.1 REQ: assignment deliberately used in a Boolean context

`string.c: 153 [2]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: the Boolean value depends on the memory model

`string.c: 198 [2]`

Informational #774: the Boolean value depends on the memory model

`string.c: 199 [2]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 225 [2]`

Informational #720, MISRA 13.1 REQ, MISRA 13.2 ADV: assignment deliberately used in a Boolean context

`string.c: 227 [2]`

Warning #533, MISRA 16.8 REQ: on this exit path, the function contains HLI only

`string.c: 230 [2]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 245 [2]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: the Boolean value depends on the memory model

`string.c: 308 [2]`

Informational #774: the Boolean value depends on the memory model

`string.c: 309 [2]`

Warning #533, MISRA 16.8 REQ: the absence of a return statement on the HLI exit path is deliberate

`string.c: 340 [2]`

Elective Note #926, MISRA 11.4 ADV: safe conversion, from 'const char *' to 'char *'

`string.c: 373 [2]`

Elective Note #926, MISRA 11.4 ADV: safe conversion, from 'const char *' to 'char *'

`string.c: 378 [2]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 384 [2]`

Elective Note #926, MISRA 11.4 ADV: safe conversion, from 'const char *' to 'char *'

`string.c: 393 [2]`

Elective Note #926, MISRA 11.4 ADV: safe conversion, from 'const char *' to 'char *'

`string.c: 395 [2]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 412 [2]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 429 [2]`

Elective Note #926, MISRA 11.4 ADV: safe conversion, from 'const char *' to 'char *'

`string.c: 435 [2]`

Elective Note #926, MISRA 11.4 ADV: safe conversion, from 'const char *' to 'char *'

`string.c: 449 [2]`

Elective Note #926, MISRA 11.4 ADV: safe conversion, from 'const char *' to 'char *'

`string.c: 465 [2]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 475 [2]`

Elective Note #926, MISRA 11.4 ADV: safe conversion, from 'const char *' to 'char *'

`string.c: 494 [2]`

Elective Note #926, MISRA 11.4 ADV: safe conversion, from 'const char *' to 'char *'

`string.c: 503 [2]`

Elective Note #960, MISRA 7.1 REQ: safe use of octal escape sequences

`terminal.c: 11 [2]`

Elective Note #960, MISRA 4.1 REQ: safe use of octal escape sequences

`terminal.c: 12 [2]`

Warning #685: the function may take an integer as argument

`terminal.c: 68 [2]`

Warning #641: use the integer model for enums

`terminal.c: 73 [2]`

Elective Note #946, MISRA 17.2 REQ, MISRA 17.3 REQ: this file implements the memory management standard library functions; the compiler compares/subtracts the addresses pointed to by the two operands

`alloc.c: 71 [2]`

Elective Note #960, MISRA 10.1 REQ: the result of sizeof() has type size_t

`alloc.c: 87 [2]`

Elective Note #960, MISRA 10.1 REQ: the result of sizeof() has type size_t

`alloc.c: 98 [2]`

Elective Note #960, MISRA 10.1 REQ: the result of sizeof() has type size_t

`alloc.c: 282 [2]`

Warning #586: accept 'free'

`alloc.c: 305 [2]`

Warning #424: deallocation is appropriate

`alloc.c: 306 [2]`

Informational #828: according to ANSI-C, setjmp must be a macro

`setjmp.h: 23 [2]`

Informational #715, MISRA 16.7 ADV: this function contains HLI only

`setjmp.c: 57 [2]`

Informational #818: this function contains HLI only

`setjmp.c: 57 [2]`

Informational #715, MISRA 16.7 ADV: this function contains HLI only

`setjmp.c: 107 [2]`

Informational #818: this function contains HLI only

`setjmp.c: 107 [2]`

Informational #766: hidef.h is used in HLI

`setjmp.c: 108 [2]`

Informational #766:

`runtime.sgm: 10 [2]`

Warning #537: allow multiple use

`runtime.sgm: 11 [2]`

Warning #451: push.sgm, non_bank.sgm and runtime.sgm are not regular header files, they contain CODE_SEG/push/pop pragmas

`dregs.h: 21 [4]`

Warning #451: default.sgm and pop.sgm are not regular header files, they contain CODE_SEG/push/pop pragmas

`dregs.h: 49 [4]`

Warning #451: non_bank.sgm and runtime.sgm are not regular header files, they contain CODE_SEG/push/pop pragmas

`dadd.c: 13 [4]`

Elective Note #957, MISRA 8.1 REQ: these are runtime support functions and, as such, are not meant to be called in user code; they are only invoked via jumps, in compiler-generated code

`dadd.c: 18 [4]`

Informational #766: header file 'dregs.h' is used, but in HLI code

`dadd.c: 448 [4]`

Warning #451: non_bank.sgm and runtime.sgm are not regular header files, they contain CODE_SEG/push/pop pragmas

`dansi.c: 15 [4]`

Elective Note #957, MISRA 8.1 REQ: these are runtime support functions and, as such, are not meant to be called in user code; they are only invoked via jumps, in compiler-generated code

`dansi.c: 23 [4]`

Informational #750: local macros referenced, but in HLI

`dansi.c: 24 [4]`

Informational #752: symbol 'modff' is used in HLI

`dansi.c: 242 [7]`

Informational #752: symbol 'modff' is referenced in HLI code

`dansi.c: 243 [7]`

Informational #752: symbol 'frexpf' is used in HLI

`dansi.c: 249 [3]`

Informational #752: symbol 'ldexpf' is used in HLI

`dansi.c: 255 [3]`

Informational #766: header file 'dregs.h' is used, but in HLI code

`dansi.c: 454 [4]`

Elective Note #957, MISRA 8.1 REQ: these are runtime support functions and, as such, are not meant to be called in user code; they are only invoked via jumps, in compiler-generated code

`dcmp.c: 15 [4]`

Informational #766: non_bank.sgm is not a regular header file, it contains a conditionally compiled CODE_SEG pragma

`dcmp.c: 84 [4]`

Warning #451: push.sgm, non_bank.sgm and runtime.sgm are not regular header files, they contain CODE_SEG/push/pop pragmas

`dconv.h: 18 [4]`

Warning #451: default.sgm and pop.sgm are not regular header files, they contain CODE_SEG/push/pop pragmas

`dconv.h: 35 [4]`

Warning #451: non_bank.sgm and runtime.sgm are not regular header files, they contain CODE_SEG/push/pop pragmas

`dconv.c: 14 [4]`

Elective Note #957, MISRA 8.1 REQ: these are runtime support functions and, as such, are not meant to be called in user code;

they are only invoked via jumps, in compiler-generated code

`dconv.c: 19 [4]`

Informational #766: header file 'dregs.h' is used, but in HLI code

`dconv.c: 482 [4]`

Warning #451: push.sgm, non_bank.sgm and runtime.sgm are not regular header files, they contain CODE_SEG/push/pop pragmas

`fregs.h: 20 [4]`

Warning #451: default.sgm and pop.sgm are not regular header files, they contain CODE_SEG/push/pop pragmas

`fregs.h: 51 [4]`

Warning #451: non_bank.sgm and runtime.sgm are not regular header files, they contain CODE_SEG/push/pop pragmas

`dfconv.c: 15 [4]`

Elective Note #957, MISRA 8.1 REQ: these are runtime support functions and, as such, are not meant to be called in user code; they are only invoked via jumps, in compiler-generated code

`dfconv.c: 20 [4]`

Elective Note #960, MISRA 19.6 REQ: macro names need to be reused across the runtime support implementation

`dfconv.c: 21 [4]`

Informational #766: header files 'dregs.h' and 'fregs.h' are used, but in HLI code

`dfconv.c: 245 [4]`

Warning #451: non_bank.sgm and runtime.sgm are not regular header files, they contain CODE_SEG/push/pop pragmas

`dmul.c: 14 [4]`

Elective Note #957, MISRA 8.1 REQ: these are runtime support functions and, as such, are not meant to be called in user code; they are only invoked via jumps, in compiler-generated code

`dmul.c: 19 [4]`

Informational #766: header file 'dregs.h' is used, but in HLI code

`dmul.c: 475 [4]`

Warning #451: non_bank.sgm and runtime.sgm are not regular header files, they contain CODE_SEG/push/pop pragmas

`dregs.c: 14 [4]`

Informational #766: non_bank.sgm is not a regular header file, it contains a conditionally compiled CODE_SEG pragma

`dregs.c: 575 [4]`

Warning #451: non_bank.sgm and runtime.sgm are not regular header files, they contain CODE_SEG/push/pop pragmas

`fadd.c: 15 [4]`

Elective Note #957, MISRA 8.1 REQ: these are runtime support functions and, as such, are not meant to be called in user code; they are only invoked via jumps, in compiler-generated code

`fadd.c: 20 [4]`

Informational #766: header files 'dregs.h' and 'fregs.h' are used, but in HLI code

`fadd.c: 315 [4]`

Warning #451: non_bank.sgm and runtime.sgm are not regular header files, they contain CODE_SEG/push/pop pragmas

`fansi.c: 16 [4]`

Elective Note #957, MISRA 8.1 REQ: these are runtime support functions and, as such, are not meant to be called in user code; they are only invoked via jumps, in compiler-generated code

`fansi.c: 24 [4]`

Warning #528: function RetErrDom is referenced in HLI

`fansi.c: 288 [4]`

Informational #766: header file 'fregs.h' are used, but in HLI code

`fansi.c: 335 [4]`

Elective Note #957, MISRA 8.1 REQ: these are runtime support functions and, as such, are not meant to be called in user code; they are only invoked via jumps, in compiler-generated code

`fcmp.c: 14 [4]`

Informational #766: non_bank.sgm is not a regular header file, it contains a conditionally compiled CODE_SEG pragma

`fcmp.c: 80 [4]`

Warning #451: non_bank.sgm and runtime.sgm are not regular header files, they contain CODE_SEG/push/pop pragmas

`fconv.c: 13 [4]`

Elective Note #957, MISRA 8.1 REQ: these are runtime support functions and, as such, are not meant to be called in user code; they are only invoked via jumps, in compiler-generated code

`fconv.c: 18 [4]`

Informational #766: header file 'fregs.h' is used, but in HLI code

`fconv.c: 277 [4]`

Warning #451: non_bank.sgm and runtime.sgm are not regular header files, they contain CODE_SEG/push/pop pragmas

`fmul.c: 15 [4]`

Elective Note #957, MISRA 8.1 REQ: these are runtime support functions and, as such, are not meant to be called in user code; they are only invoked via jumps, in compiler-generated code

`fmul.c: 20 [4]`

Informational #766: header file 'fregs.h' are used, but in HLI code

`fmul.c: 392 [4]`

Warning #451: non_bank.sgm and runtime.sgm are not regular header files, they contain CODE_SEG/push/pop pragmas

`fregs.c: 16 [4]`

Informational #766: non_bank.sgm is not a regular header file, it contains a conditionally compiled CODE_SEG pragma

`fregs.c: 435 [4]`

Warning #451: non_bank.sgm and runtime.sgm are not regular header files, they contain CODE_SEG/push/pop pragmas

`lansi.c: 10 [4]`

Informational #766: non_bank.sgm is not a regular header file, it contains a conditionally compiled CODE_SEG pragma

`lansi.c: 90 [4]`

Elective Note #957, MISRA 8.1 REQ: these are runtime support functions and, as such, are not meant to be called in user code; they are only invoked via jumps, in compiler-generated code

`rtshc12.c: 15 [2]`

Elective Note #960, MISRA 10.5 REQ: '<<' applied to an operand of type 'unsigned int' - which is not a sub-integer type

`rtshc12.c: 19 [2]`

Informational #750: the macro is referenced in HLI

`rtshc12.c: 995 [2]`

Informational #750: the macro is referenced in HLI

`rtshc12.c: 1006 [2]`

Informational #766: runtime.sgm is not a regular header file, it contains a CODE_SEG pragma

`rtshc12.c: 1907 [2]`

Warning #451: non_bank.sgm and runtime.sgm are not regular header files, they contain CODE_SEG/push/pop pragmas

`vregs.c: 15 [4]`

Informational #752: symbol '_NEG_P' is used in HLI

`vregs.c: 55 [4]`

Informational #766: runtime.sgm is not a regular header file, it contains a conditionally compiled CODE_SEG pragma

`vregs.c: 171 [4]`

Informational #708: initialization applied to the first named member of the union

`math.c: 83 [8]`

Informational #708: initialization applied to the first named member of the union

`math.c: 84 [8]`

Informational #708: initialization applied to the first named member of the union

`mathf.c: 177 [8]`

Informational #708: initialization applied to the first named member of the union

`mathf.c: 178 [8]`

Informational #777: the purpose of the test is to compare the bit patterns for an exact match

`mathf.c: 236 [8]`

Informational #704: y is positive

`mathf.c: 291 [8]`

Elective Note #960: y is positive

`mathf.c: 291 [8]`

Informational #777: the purpose of the test is to compare the bit patterns for an exact match

`mathf.c: 791 [8]`

Informational #777: the purpose of the test is to compare the bit patterns for an exact match

`mathf.c: 804 [8]`

Informational #777: the purpose of the test is to compare the bit patterns for an exact match

`mathf.c: 806 [8]`

Informational #777: the purpose of the test is to compare the bit patterns for an exact match

`mathf.c: 825 [8]`

Informational #777: the purpose of the test is to compare the bit patterns for an exact match

`mathf.c: 826 [8]`

Informational #777: the purpose of the test is to compare the bit patterns for an exact match

`mathf.c: 832 [8]`

Informational #777: the purpose of the test is to compare the bit patterns for an exact match

`mathf.c: 1113 [8]`

Informational #777: the purpose of the test is to compare the bit patterns for an exact match

`mathf.c: 1125 [8]`

Informational #826: safe conversion

`scanf.c: 403 [8]`

Informational #826: safe conversion

`scanf.c: 405 [8]`

Warning #528: symbol 'D_LDEXP' is referenced in HLI

`dansi.c: 209 [8]`

Warning #528: symbol 'RetErrDom' is referenced in HLI

`dansi.c: 399 [8]`

Elective Note #961, MISRA 19.1 ADV: non_bank.sgm and default.sgm each contain a conditionally compiled CODE_SEG

`pragma`

`start12.c: 66 [9]`

Warning #451: default.sgm contains a conditionally compiled CODE_SEG pragma

`start12.c: 92 [10]`

Warning #505: asm code

`start12.c: 191 [11]`

Warning #522: asm code

`start12.c: 191 [11]`

Warning #505: asm code

`start12.c: 263 [11]`

Warning #522: asm code

`start12.c: 263 [11]`

Warning #451: non_bank.sgm contains a conditionally compiled CODE_SEG pragma

`start12.c: 376 [9]`

Elective Note #960, MISRA 14.3 REQ: macro INIT_SP_FROM_STARTUP_DESC() expands to HLI code

`start12.c: 413 [9]`

Warning #522, MISRA 14.2 REQ: macro INIT_SP_FROM_STARTUP_DESC() expands to HLI code

`start12.c: 414 [9]`

Warning #522, MISRA 14.2 REQ: function Init() contains HLI only

`start12.c: 467 [9]`

Informational #766: non_bank.sgm is not a regular header file, it contains a conditionally compiled CODE_SEG pragma

`start12.c: 480 [9]`

Warning #451: non_bank.sgm is not a regular header files, it contains a CODE_SEG pragma

`setjmp.c: 14 [12]`

Informational #752: symbol '_SET_PAGE' is used in HLI

`setjmp.c: 18 [12]`

Informational #752: symbol '_modff' is used in HLI

`dansi.c: 239 [13]`

Warning #451: non_bank.sgm contains a conditionally compiled CODE_SEG pragma

`start12.c: 80 [14]`

Informational #752: symbol '_SET_PAGE' is referenced in HLI

`start12.c: 87 [14]`

Warning #505: asm code

`start12.c: 172 [15]`

Warning #522: asm code

`start12.c: 172 [15]`

Warning #505: asm code

`start12.c: 244 [15]`

Warning #522: asm code

`start12.c: 244 [15]`

3

XGATE

This chapter contains these topics for XGATE:

- Inline Assembly
- General Exceptions
- Per-project Exceptions

Inline Assembly

Inline assembly is altogether ignored when checking for MISRA-C:2004 compliancy.

General Exceptions

Table 1. lists the exceptions to MISRA-C:2004 rules that apply across all the library projects.

Table 1. XGATE general library exceptions to MISRA-C:2004 rules

MISRA-C:2004 Rule	Exception
6.3 ADV	inhibit the message on the use of a modifier or a type outside of a typedef
19.7 ADV	allow function-like macros
19.15 REQ	allow repeatedly included header files - all the libray headers are guarded using macros
14.7 REQ	allow multiple exit points for functions
14.5 REQ	allow 'continue' statements
18.4 REQ	allow unions
20.4 REQ, 205. REQ	accept several deprecated symbols

Per-project Exceptions

Elective Note #961, MISRA 19.3 ADV: '#' is used in HLI as an operator - see pragma NO_STRING_CONSTR above

`hidef.h: 51 [16]`

Elective Note #960, MISRA 19.6 REQ: pointer qualifiers 'far' and 'near' are not supported on XGATE

`hidef.h: 59 [16]`

Warning #683: inhibit warning on standard function being #define'd

`stdlib.h: 45 [16]`

Elective Note #960, MISRA 16.3 REQ: message reported not for a function, but for a function pointer

`stdlib.h: 82  [16]`

Elective Note #960, MISRA 16.3 REQ: message reported not for a function, but for a function pointer

`stdlib.h: 85 [16]`

Informational #715: name not used

`assert.c: 21 [16]`

Informational #766: hidef.h contains conditionally compiled code

`assert.c: 24 [16]`

Elective Note #960, MISRA 10.1 REQ: the conversion has no impact on bit pattern intepretation

`ctype.c: 11 [16]`

Elective Note #960, MISRA 19.6 REQ: character classification macros must be undefined prior to defining the corresponding character classification functions

`ctype.c: 276 [16]`

Informational #766: hidef.h contains conditionally compiled code

`ctype.c: 368 [16]`

Elective Note #960, MISRA 10.1 REQ: the result of sizeof() has type size_t

`heap.c: 20 [16]`

Warning #414: division by zero!

`heap.c: 32 [16]`

Warning #564: division by zero!

`heap.c: 32 [16]`

Elective Note #931: division by zero!

`heap.c: 32 [16]`

Informational #818, MISRA 16.7 ADV: symbol element not referenced

`heap.c: 33 [16]`

Informational #715: symbol element not referenced

`heap.c: 33 [16]`

Warning #438: symbol element not referenced

`heap.c: 33 [16]`

Informational #773: va_end is never used as an expression operand

`stdarg.h: 120 [16]`

Warning #683: inhibit warning on standard function being #define'd

`stdio.h: 76 [16]`

Elective Note #960, MISRA 16.1 REQ: standard library function implementation

`stdio.h: 149 [16]`

Elective Note #960, MISRA 16.1 REQ: standard library function implementation

`stdio.h: 150 [16]`

Elective Note #960, MISRA 16.1 REQ: standard library function implementation

`stdio.h: 157 [16]`

Elective Note #960, MISRA 16.1 REQ: standard library function implementation

`stdio.h: 158 [16]`

Elective Note #960, MISRA 16.1 REQ: standard library function implementation

`stdio.h: 160 [16]`

Elective Note #960, MISRA 16.3 REQ: message reported for a function pointer parameter

`stdio.h: 163 [16]`

Elective Note #960, MISRA 16.1 REQ: standard library function implementation

`stdio.h: 204 [16]`

Elective Note #960, MISRA 16.1 REQ: standard library function implementation

`embedded.c: 31 [16]`

Warning #643: misleading warning ('&format' does not have a far type)

`embedded.c: 36 [16]`

Elective Note #928, MISRA 11.4 ADV: safe conversion to 'char *'

`embedded.c: 37 [16]`

Warning #438: 'va_end' must be invoked before return in a variadic function

`embedded.c: 41 [16]`

Elective Note #960, MISRA 16.1 REQ: standard library function implementation

`embedded.c: 44  [16]`

Warning #643: misleading warning ('&format' does not have a far type)

`embedded.c: 49  [16]`

Elective Note #928, MISRA 11.4 ADV: safe conversion to 'char *'

`embedded.c: 50  [16]`

Warning #438: 'va_end' must be invoked before return in a variadic function

`embedded.c: 55  [16]`

Warning #625: accept unusual type modifier

`embedded.c: 64  [16]`

Elective Note #923, MISRA 11.3 ADV: no support for multiple file descriptors

`embedded.c: 91 [16]`

Informational #715: this is the implementation of a standard library function, so its prototype is left unchanged

`embedded.c: 92 [16]`

Elective Note #926, MISRA 11.4 ADV: the conversion is safe

`embedded.c: 95 [16]`

Informational #715, MISRA 16.7 ADV: standard library function implementation

`embedded.c: 97 [16]`

Informational #818: standard library function implementation

`embedded.c: 97 [16]`

Elective Note #960, MISRA 16.1 REQ: standard library function implementation

`embedded.c: 103 [16]`

Warning #643: misleading warning ('&format' does not have a far pointer)

`embedded.c: 108 [16]`

Elective Note #928, MISRA 11.4 ADV: safe conversion to 'char *'

`embedded.c: 109 [16]`

Warning #438: 'va_end' must be invoked before return in a variadic function

`embedded.c: 113 [16]`

Informational #715, MISRA 16.7 ADV: standard library function implementation

`embedded.c: 114 [16]`

Informational #818: standard library function implementation

`embedded.c: 114 [16]`

Informational #715, MISRA 16.7 ADV: standard library function implementation

`embedded.c: 119 [16]`

Informational #818: standard library function implementation

`embedded.c: 119 [16]`

Informational #715, MISRA 16.7 ADV: standard library function implementation

`embedded.c: 124 [16]`

Informational #818: standard library function implementation

`embedded.c: 124 [16]`

Informational #715, MISRA 16.7 ADV: standard library function implementation

`embedded.c: 128 [16]`

Informational #818: standard library function implementation

`embedded.c: 128 [16]`

Informational #715: this is the implementation of a standard library function, so its prototype is left unchanged

`embedded.c: 132 [16]`

Warning #511: pointer size depends on both the target and the memory model

`embedded.c: 152 [16]`

Elective Note #923, MISRA 11.3 ADV: safe conversion

`embedded.c: 153 [16]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: the boolean operation results from the expansion of macro 'CONVERT_TO_PAGED'

`embedded.c: 154 [16]`

Elective Note #960, MISRA 12.7 REQ: the signed quantity is always positive

`embedded.c: 155 [16]`

Elective Note #960, MISRA 14.3 REQ: the macro expands to HLI several statements

`embedded.c: 165 [16]`

Elective Note #960, MISRA 14.3 REQ: the macro expands to HLI several statements

`embedded.c: 169 [16]`

Informational #715: this is the implementation of a standard library function, so its prototype is left unchanged

`embedded.c: 170 [16]`

Informational #766: hidef.h contains conditionally compiled code

`embedded.c: 172 [16]`

Informational #715: inhibit message on 'a' and 'p' not being referenced

`locale.c: 17 [16]`

Elective Note #960, MISRA 7.1 REQ: safe use of octal escape sequences

`terminal.c: 11 [16]`

Elective Note #960, MISRA 4.1 REQ: safe use of octal escape sequences

`terminal.c: 12 [16]`

Warning #685: the function may take an integer as argument

`terminal.c: 68 [16]`

Warning #641: use the integer model for enums

`terminal.c: 73 [16]`

Elective Note #960, MISRA 19.6 REQ: #undef required to support non-ANSI pointer qualifiers 'near' and 'far'

`non_bank.sgm: 1 [16]`

Warning #537: allow multiple use

`non_bank.sgm: 24 [16]`

Elective Note #961, MISRA 19.13 ADV: '#' is used in HLI as an operator - see pragma NO_STRING_CONSTR above

`signal.c: 17 [16]`

Elective Note #960, MISRA 14.3 REQ: macro HALTX expands to several HLI statements

`signal.c: 28 [16]`

Elective Note #960, MISRA 19.6 REQ: #undef required to support non-ANSI pointer qualifiers 'near' and 'far'

`default.sgm: 1 [16]`

Warning #537: allow multiple use

`default.sgm: 24 [16]`

Elective Note #923, MISRA 11.3 ADV: safe cast (no truncation) because the maximum signal number is 23 (SIGALRM)

`signal.c: 38 [16]`

Elective Note #923, MISRA 11.3 ADV: safe casts

`signal.c: 39 [16]`

Elective Note #929, MISRA 11.4 ADV: safe casts

`signal.c: 39 [16]`

Elective Note #923, MISRA 11.3 ADV: safe casts

`signal.c: 45 [16]`

Elective Note #929, MISRA 11.4 ADV: safe casts

`signal.c: 45 [16]`

Elective Note #960, MISRA 14.3 REQ: macro HALTX expands to several HLI statements

`signal.c: 54 [16]`

Informational #715: the function contains inline assembly only

`signal.c: 56 [16]`

Informational #766: non_bank.sgm is not a regular header file, it contains a conditionally CODE_SEG pragma

`signal.c: 61 [16]`

Elective Note #946, MISRA 17.2 REQ, MISRA 17.3 REQ: this file implements the memory management standard library functions; the compiler compares/subtracts the addresses pointed to by the two operands

`alloc.c: 71 [16]`

Elective Note #960, MISRA 10.1 REQ: the result of sizeof() has type size_t

`alloc.c: 87 [16]`

Elective Note #960, MISRA 10.1 REQ: the result of sizeof() has type size_t

`alloc.c: 98 [16]`

Elective Note #960, MISRA 10.1 REQ: the result of sizeof() has type size_t

`alloc.c: 282 [16]`

Warning #586: accept 'free'

`alloc.c: 305 [16]`

Warning #424: deallocation is appropriate

`alloc.c: 306 [16]`

Elective Note #960, MISRA 16.3 REQ: this is a function pointer declaration

`printf.c: 83 [16]`

Elective Note #960, MISRA 7.1 REQ: safe use of octal constants

`printf.c: 103 [16]`

Warning #625: (accept unusual type modifier; no precision loss)

`printf.c: 123 [17]`

Warning #619: (accept unusual type modifier; no precision loss)

`printf.c: 123 [17]`

Warning #619: no precision loss

`printf.c: 128 [17]`

Informational #702, MISRA 12.7 REQ: allow signed right shift, its positive anyway

`printf.c: 137 [17]`

Elective Note #960: allow signed right shift, its positive anyway

`printf.c: 137 [17]`

Informational #702, MISRA 12.7 REQ: allow signed right shift, its positive anyway

`printf.c: 150 [17]`

Elective Note #960: allow signed right shift, its positive anyway

`printf.c: 150 [17]`

Informational #750: suppress the message on macros 'DIGITS' and 'BOUND' not being referenced

`printf.c: 167 [16]`

Elective Note #960, MISRA 10.5 REQ: '~' applied to an operand of type 'unsigned int'

`printf.c: 322 [16]`

Elective Note #960, MISRA 10.5 REQ: '~' applied to an operand of type 'unsigned int'

`printf.c: 325 [16]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`printf.c: 330 [16]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`printf.c: 346 [16]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`printf.c: 400 [16]`

Informational #826: safe conversion

`printf.c: 401 [16]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`printf.c: 405 [16]`

Informational #826: safe conversion

`printf.c: 406 [16]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`printf.c: 410 [16]`

Informational #826: safe conversion

`printf.c: 411 [16]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`printf.c: 419 [16]`

Elective Note #926, MISRA 11.4 ADV: conversion is necessary and safe

`printf.c: 426 [16]`

Informational #801: Use of goto is not deprecated

`printf.c: 428 [16]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`printf.c: 447 [16]`

Warning #613: Possible use of null pointer 'str' in argument

`printf.c: 478 [16]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: the Boolean value is target-dependent

`printf.c: 493 [16]`

Informational #774: the Boolean value is target-dependent

`printf.c: 493 [16]`

Informational #801: Use of goto is not deprecated

`printf.c: 505 [16]`

Elective Note #960, MISRA 10.5 REQ: '~' applied to an operand of type 'unsigned int'

`printf.c: 516 [16]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`printf.c: 523 [16]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`printf.c: 528 [16]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: target-dependent Boolean expressions

`printf.c: 533 [16]`

Informational #774: target-dependent Boolean expressions

`printf.c: 533 [16]`

Informational #845: target-dependent Boolean expressions

`printf.c: 533 [16]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`printf.c: 535 [16]`

Elective Note #923, MISRA 11.3 ADV: the cast is necessary, see comment above

`printf.c: 536 [16]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`printf.c: 539 [16]`

Informational #801: Use of goto is not deprecated

`printf.c: 547 [16]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`printf.c: 553 [16]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`printf.c: 558 [16]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`printf.c: 563 [16]`

Elective Note #960: the result of sizeof() is size_t, which is defined to 'unsigned int'

`printf.c: 579 [16]`

Elective Note #960: the result of sizeof() is size_t, which is defined to 'unsigned int'

`printf.c: 581 [16]`

Elective Note #960: the result of sizeof() is size_t, which is defined to 'unsigned int'

`printf.c: 583 [16]`

Warning #661: (no out-of-bounds access)

`printf.c: 714 [16]`

Warning #662: (no out-of-bounds access)

`printf.c: 714 [16]`

Warning #661: (no out-of-bounds access)

`printf.c: 716 [16]`

Informational #825: fallthrough is deliberate

`printf.c: 764 [17]`

Informational #801: Use of goto is not deprecated

`printf.c: 766 [17]`

Informational #801: Use of goto is not deprecated

`printf.c: 770 [17]`

Informational #825: fall through is deliberate

`printf.c: 775 [17]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`printf.c: 782 [17]`

Elective Note #960: the type of the result of sizeof() is size_t, which is defined to 'unsigned int'

`printf.c: 815 [17]`

Elective Note #960: the type of the result of sizeof() is size_t, which is defined to 'unsigned int'

`printf.c: 825 [17]`

Elective Note #960: the type of the result of sizeof() is size_t, which is defined to 'unsigned int'

`printf.c: 826 [17]`

Elective Note #960, MISRA 10.5 REQ: '~' applied to an operand of type 'unsigned int'

`printf.c: 842 [17]`

Elective Note #960: the type of the result of sizeof() is size_t, which is defined to 'unsigned int'

`printf.c: 874 [17]`

Elective Note #960: the type of the result of sizeof() is size_t, which is defined to 'unsigned int'

`printf.c: 875 [17]`

Elective Note #960, MISRA 10.5 REQ: '~' applied to operand of type 'unsigned int')

`printf.c: 885 [17]`

Elective Note #960: the type of the result of sizeof() is size_t, which is defined to 'unsigned int'

`printf.c: 888 [17]`

Elective Note #960: the type of the result of sizeof() is size_t, which is defined to 'unsigned int'

`printf.c: 898 [17]`

Elective Note #960: the type of the result of sizeof() is size_t, which is defined to 'unsigned int'

`printf.c: 900 [17]`

Elective Note #960: the type of the result of sizeof() is size_t, which is defined to 'unsigned int'

`printf.c: 919 [17]`

Elective Note #960: the type of the result of sizeof() is size_t, which is defined to 'unsigned int'

`printf.c: 991 [17]`

Elective Note #960: the type of the result of sizeof() is size_t, which is defined to 'unsigned int'

`printf.c: 992 [17]`

Informational #825: fallthrough is deliberate

`printf.c: 1066 [16]`

Informational #818, MISRA 16.7 ADV: this is a standard library function, cannot change its prototype

`printf.c: 1073 [16]`

Elective Note #960, MISRA 16.3 REQ: this is a function pointer parameter

`printf.c: 1149 [16]`

Elective Note #960, MISRA 16.1 REQ: standard library function implementation

`printf.c: 1190 [16]`

Warning #643: misleading warning ('&format' does not have a far pointer type)

`printf.c: 1196 [16]`

Elective Note #928, MISRA 11.4 ADV: safe conversion to 'char *'

`printf.c: 1197 [16]`

Warning #438: 'va_end' must be invoked before return in a variadic function

`printf.c: 1202 [16]`

Informational #766: header file contains conditionally compiled code

`printf.c: 1215 [16]`

Warning #625: accept unusual type modifier

`scanf.c: 104 [16]`

Warning #625: accept unusual type modifier

`scanf.c: 154 [16]`

Warning #625: accept unusual type modifier

`scanf.c: 157 [16]`

Warning #625: accept unusual type modifier

`scanf.c: 161 [16]`

Informational #801: Use of goto is not deprecated

`scanf.c: 184 [16]`

Elective Note #931: the expression is safe

`scanf.c: 191 [16]`

Informational #801: Use of goto is not deprecated

`scanf.c: 195 [16]`

Informational #801: Use of goto is not deprecated

`scanf.c: 199 [16]`

Informational #801: Use of goto is not deprecated

`scanf.c: 228 [16]`

Informational #801: Use of goto is not deprecated

`scanf.c: 237 [16]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`scanf.c: 262 [16]`

Informational #826: safe conversion

`scanf.c: 263 [16]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`scanf.c: 268 [16]`

Informational #826: safe conversion

`scanf.c: 269 [16]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`scanf.c: 273 [16]`

Informational #826: safe conversion

`scanf.c: 274 [16]`

Informational #801: Use of goto is not deprecated

`scanf.c: 286 [16]`

Informational #801: Use of goto is not deprecated

`scanf.c: 294 [16]`

Informational #801: Use of goto is not deprecated

`scanf.c: 305 [16]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`scanf.c: 338 [16]`

Informational #826: safe conversion

`scanf.c: 339 [16]`

Elective Note #961, MISRA 17.5 ADV: multiple indirection levels necessary in order to implement support for '%p'

`scanf.c: 343 [16]`

Warning #511: pointer size depends on both the target and the memory model (if truncation occurs, it is expected)

`scanf.c: 344 [16]`

Elective Note #923: pointer size depends on both the target and the memory model (if truncation occurs, it is expected)

`scanf.c: 344 [16]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`scanf.c: 345 [16]`

Informational #826: safe conversion

`scanf.c: 346 [16]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`scanf.c: 360 [16]`

Informational #826: safe conversion

`scanf.c: 361 [16]`

Informational #801: Use of goto is not deprecated

`scanf.c: 379 [17]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`scanf.c: 398 [17]`

Informational #826: safe conversion

`scanf.c: 399 [17]`

Warning #539: conditionally compiled 'if' clause

`scanf.c: 413 [17]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`scanf.c: 436 [16]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`scanf.c: 481 [16]`

Elective Note #946, MISRA 17.2 REQ, MISRA 17.3 REQ: the two pointers point into the same array object

`scanf.c: 521 [16]`

Elective Note #947: the two pointers point into the same array object

`scanf.c: 521 [16]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`scanf.c: 525 [16]`

Informational #826: safe conversion

`scanf.c: 526 [16]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`scanf.c: 530 [16]`

Informational #826: safe conversion

`scanf.c: 531 [16]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`scanf.c: 535 [16]`

Informational #826: safe conversion

`scanf.c: 536 [16]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`scanf.c: 555 [16]`

Elective Note #929, MISRA 11.4 ADV: conversion is safe

`scanf.c: 569 [16]`

Warning #539: conditionally compiled 'if' clause

`scanf.c: 570 [16]`

Informational #818, MISRA 16.7 ADV: standard library function, cannot change its prototype

`scanf.c: 592 [16]`

Elective Note #960, MISRA 16.1 REQ: standard library function implementation

`scanf.c: 597 [16]`

Warning #643: misleading warning ('&format' does not have a far pointer type)

`scanf.c: 601 [16]`

Elective Note #928, MISRA 11.4 ADV: safe conversion to 'char *'

`scanf.c: 602 [16]`

Warning #438: 'va_end' must be invoked before return in a variadic function

`scanf.c: 606 [16]`

Elective Note #961, MISRA 19.13 ADV: '#' used not as the stringification preprocessing operator, but as an inline assembly operator

`stdlib.c: 25 [16]`

Elective Note #960, MISRA 14.3 REQ: the macro expands to several HLI statements

`stdlib.c: 73 [16]`

Informational #715: standard library function implementation

`stdlib.c: 75 [16]`

Elective Note #960, MISRA 14.3 REQ: the macro expands to several HLI statements

`stdlib.c: 79 [16]`

Informational #715: standard library function implementation

`stdlib.c: 81 [16]`

Elective Note #960, MISRA 14.3 REQ: the macro expands to several HLI statements

`stdlib.c: 85 [16]`

Informational #715: standard library function implementation

`stdlib.c: 87 [16]`

Elective Note #960, MISRA 14.3 REQ: the macro expands to several HLI statements

`stdlib.c: 91 [16]`

Informational #715: standard library function implementation

`stdlib.c: 93 [16]`

Elective Note #960, MISRA 14.3 REQ: the macro expands to several HLI statements

`stdlib.c: 97 [16]`

Informational #715, MISRA 16.7 ADV: standard library function implementation

`stdlib.c: 99 [16]`

Informational #818: standard library function implementation

`stdlib.c: 99 [16]`

Elective Note #960, MISRA 14.3 REQ: the macro expands to several HLI statements

`stdlib.c: 103 [16]`

Informational #715, MISRA 16.7 ADV: standard library function implementation

`stdlib.c: 105 [16]`

Informational #818: standard library function implementation

`stdlib.c: 105 [16]`

Elective Note #960, MISRA 14.3 REQ: the macro expands to several HLI statements

`stdlib.c: 109 [16]`

Informational #715, MISRA 16.7 ADV: standard library function implementation

`stdlib.c: 111 [16]`

Informational #818: standard library function implementation

`stdlib.c: 111 [16]`

Elective Note #960, MISRA 14.3 REQ: the macro expands to several HLI statements

`stdlib.c: 115 [16]`

Informational #715, MISRA 16.7 ADV: standard library function implementation

`stdlib.c: 117 [16]`

Informational #818: standard library function implementation

`stdlib.c: 117 [16]`

Warning #625: accept unusual type modifier

`stdlib.c: 145 [16]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 156 [16]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 162 [16]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 171 [16]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 175 [16]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 186 [16]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 190 [16]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 207 [16]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 211 [16]`

Elective Note #960, MISRA 14.3 REQ: the macro does not always expand to a null statement

`stdlib.c: 215 [16]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 219 [16]`

Elective Note #926, MISRA 11.4 ADV: safe cast

`stdlib.c: 225 [16]`

Warning #625: accept unusual type modifier

`stdlib.c: 246 [16]`

Warning #610: pointer tested against NULL

`stdlib.c: 252 [16]`

Elective Note #926, MISRA 11.4 ADV: safe cast

`stdlib.c: 253 [16]`

Warning #610: pointer tested against NULL

`stdlib.c: 263 [16]`

Elective Note #926, MISRA 11.4 ADV: safe cast

`stdlib.c: 264 [16]`

Elective Note #946, MISRA 17.2 REQ, MISRA 17.3 REQ: the two pointers point into the same array object

`stdlib.c: 272 [16]`

Elective Note #947: the two pointers point into the same array object

`stdlib.c: 272 [16]`

Warning #610: pointer tested against NULL

`stdlib.c: 278 [16]`

Elective Note #926, MISRA 11.4 ADV: safe cast

`stdlib.c: 279 [16]`

Warning #625: accept unusual type modifier

`stdlib.c: 302 [17]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 319 [17]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 326 [17]`

Elective Note #960, MISRA 14.3 REQ: the macro does not always expand to a null statement

`stdlib.c: 330 [17]`

Elective Note #960, MISRA 14.3 REQ: the macro does not always expand to a null statement

`stdlib.c: 332 [17]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 338 [17]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 341 [17]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 345 [17]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 347 [17]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 350 [17]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 365 [17]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 389 [17]`

Elective Note #960, MISRA 14.3 REQ: the macro does not always expand to a null statement

`stdlib.c: 395 [17]`

Elective Note #960, MISRA 14.3 REQ: the macro does not always expand to a null statement

`stdlib.c: 403 [17]`

Informational #801: Use of goto is not deprecated

`stdlib.c: 433 [17]`

Elective Note #960, MISRA 14.3 REQ: the macro does not always expand to a null statement

`stdlib.c: 440 [17]`

Elective Note #960, MISRA 14.3 REQ: the macro does not always expand to a null statement

`stdlib.c: 442 [17]`

Elective Note #960, MISRA 14.3 REQ: the macro does not always expand to a null statement

`stdlib.c: 449 [17]`

Elective Note #960, MISRA 14.3 REQ: the macro does not always expand to a null statement

`stdlib.c: 451 [17]`

Elective Note #960, MISRA 14.3 REQ: the macro does not always expand to a null statement

`stdlib.c: 458 [17]`

Elective Note #960, MISRA 14.3 REQ: the macro does not always expand to a null statement

`stdlib.c: 460 [17]`

Elective Note #926, MISRA 11.4 ADV: safe cast

`stdlib.c: 477 [17]`

Warning #625: accept unusual type modifier

`stdlib.c: 539 [16]`

Elective Note #960, MISRA 16.3 REQ: function pointer parameter

`stdlib.c: 610 [16]`

Warning #625: accept unusual type modifier

`stdlib.c: 614 [16]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`stdlib.c: 645 [16]`

Elective Note #927, MISRA 11.4 ADV: deliberate cast from 'char*' to a word pointer type

`stdlib.c: 648 [16]`

Informational #826: deliberate cast from 'char *' to a word pointer type

`stdlib.c: 649 [16]`

Elective Note #960, MISRA 16.3 REQ: function pointer parameter

`stdlib.c: 667 [16]`

Warning #625: accept unusual type modifier

`stdlib.c: 670 [16]`

Elective Note #931: safe expression

`stdlib.c: 678 [16]`

Elective Note #960, MISRA 12.4 REQ: safe expression

`stdlib.c: 680 [16]`

Informational #766: header files 'math.h' and 'float.h' contain conditionally compiled code

`stdlib.c: 778 [16]`

Elective Note #926, MISRA 11.4 ADV: deliberate cast

`string.c: 17 [16]`

Informational #715: implementation of a standard library function

`string.c: 19 [16]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 22 [16]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 34 [16]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 35 [16]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 58 [16]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 59 [16]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 76 [16]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 77 [16]`

Elective Note #946, MISRA 17.2 REQ, MISRA 17.3 REQ: memmove implementation : need to establish if the two memory areas overlap

`string.c: 79 [16]`

Elective Note #946, MISRA 17.2 REQ, MISRA 17.3 REQ: memmove implementation : need to establish if the two memory areas overlap

`string.c: 83 [16]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 95 [16]`

Elective Note #926, MISRA 11.4 ADV: safe conversion to 'char *'

`string.c: 118 [16]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 119 [16]`

Elective Note #946, MISRA 17.2 REQ, MISRA 17.3 REQ: the two pointers point into the same string

`string.c: 124 [16]`

Elective Note #947, MISRA 17.2 REQ: the two pointers point into the same string

`string.c: 125 [16]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 131 [16]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 140 [16]`

Informational #720, MISRA 13.1 REQ, MISRA 13.2 ADV: assignment deliberately used in a Boolean context

`string.c: 143 [16]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 148 [16]`

Informational #820, MISRA 13.1 REQ: assignment deliberately used in a Boolean context

`string.c: 153 [16]`

Informational #715: this function contains HLI only

`string.c: 172 [16]`

Informational #715: this function contains HLI only

`string.c: 173 [16]`

Informational #818, MISRA 16.7 ADV: this function contains HLI only

`string.c: 174 [16]`

Warning #533, MISRA 16.8 REQ: this function contains HLI only

`string.c: 187 [16]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 245 [16]`

Informational #715: this function contains HLI only

`string.c: 273 [16]`

Informational #715: this function contains HLI only

`string.c: 274 [16]`

Warning #533, MISRA 16.8 REQ: this function contains HLI only

`string.c: 299 [16]`

Elective Note #926, MISRA 11.4 ADV: safe conversion, from 'const char *' to 'char *'

`string.c: 373 [16]`

Elective Note #926, MISRA 11.4 ADV: safe conversion, from 'const char *' to 'char *'

`string.c: 378 [16]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 384 [16]`

Elective Note #926, MISRA 11.4 ADV: safe conversion, from 'const char *' to 'char *'

`string.c: 393 [16]`

Elective Note #926, MISRA 11.4 ADV: safe conversion, from 'const char *' to 'char *'

`string.c: 395 [16]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 412 [16]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 429 [16]`

Elective Note #926, MISRA 11.4 ADV: safe conversion, from 'const char *' to 'char *'

`string.c: 435 [16]`

Elective Note #926, MISRA 11.4 ADV: safe conversion, from 'const char *' to 'char *'

`string.c: 449 [16]`

Elective Note #926, MISRA 11.4 ADV: safe conversion, from 'const char *' to 'char *'

`string.c: 465 [16]`

Warning #625: options ConstQualiNear and -NonConstQualiNear force qualifier 'far' on library pointer types

`string.c: 475 [16]`

Elective Note #926, MISRA 11.4 ADV: safe conversion, from 'const char *' to 'char *'

`string.c: 494 [16]`

Elective Note #926, MISRA 11.4 ADV: safe conversion, from 'const char *' to 'char *'

`string.c: 503 [16]`

Informational #708: initialization applied to the first named member of the union

`math.c: 73 [17]`

Informational #708: initialization applied to the first named member of the union

`math.c: 77 [17]`

Informational #777: the purpose of the test is to compare the bit patterns for an exact match

`math.c: 185 [17]`

Elective Note #934: no dynamic linking, an absolute address is obtained

`math.c: 494 [17]`

Elective Note #960, MISRA 12.4 REQ: no impact if 'fabs' is not called during expression evaluation

`math.c: 495 [17]`

Informational #750: suppress the messages on several local macros not being referenced

`math.c: 804 [17]`

Informational #777: the purpose of the test is to compare the bit patterns for an exact match

`math.c: 1205 [17]`

Informational #777: the purpose of the test is to compare the bit patterns for an exact match

`math.c: 1215 [17]`

Informational #766: hidef.h contains conditionally compiled code

`math.c: 1234 [17]`

Informational #766: hidef.h contains conditionally compiled code

`mathf.c: 1145 [17]`

Elective Note #957, MISRA 8.1 REQ: these are runtime support functions and, as such, are not meant to be called in user code; they are only invoked via jumps, in compiler-generated code

`rtsxgate.cxgate: 12 [17]`

Informational #753: local enums used in HLI

`rtsxgate.cxgate: 13 [17]`

Informational #749: local enumeration constants used in HLI

`rtsxgate.cxgate: 14 [17]`

Informational #715: this function contains HLI only

`rtsxgate.cxgate: 1226 [17]`

Informational #715: this function contains HLI only

`rtsxgate.cxgate: 1227 [17]`

Informational #818, MISRA 16.7 ADV: this function contains HLI only

`rtsxgate.cxgate: 1228 [17]`

Warning #533, MISRA 16.8 REQ: this function contains HLI only

`rtsxgate.cxgate: 1274 [17]`

Informational #715: this function contains HLI only

`rtsxgate.cxgate: 1283 [17]`

Informational #715: this function contains HLI only

`rtsxgate.cxgate: 1284 [17]`

Warning #533, MISRA 16.8 REQ: this function contains HLI only

`rtsxgate.cxgate: 1334 [17]`

Informational #715: this function contains HLI only

`rtsxgate.cxgate: 1342 [17]`

Informational #715: this function contains HLI only

`rtsxgate.cxgate: 1343 [17]`

Informational #818, MISRA 16.7 ADV: this function contains HLI only

`rtsxgate.cxgate: 1344 [17]`

Warning #533, MISRA 16.8 REQ: this function contains HLI only

`rtsxgate.cxgate: 1391 [17]`

Informational #766: hidef.h contains CODE_SEG, CONST_SEG and STRING_SEG pragmas for XGATE

`rtsxgate.cxgate: 1397 [17]`

Warning #506, MISRA 13.7 REQ, MISRA 14.1 REQ: default data type formats can be changed with the -T option

`printf.c: 1062 [18]`

Warning #522, MISRA 14.2 REQ: the argument is dropped

`printf.c: 1063 [18]`

References

This appendix contains the list of targets for HCS12 and XGATE:

1. Valid on all targets
2. Valid on all targets except: `config-c_mb_startup.lnt, config-c_ml_startup.lnt, config-c_ms_startup.lnt, config-hcs12x_c_mb_startup.lnt, confighcs12x_c_mb_startup_no_jsr2bsr.lnt, config-hcs12x_c_ml_startup.lnt, config-hcs12x_c_ml_startup_const_near.lnt, config-hcs12x_c_ml_startup_no_jsr2bsr.lnt, confighcs12x_c_ml_startup_non_const_near.lnt, config-hcs12x_c_ms_startup.lnt, config-hcs12x_c_ms_startup_no_jsr2bsr.lnt`
3. Valid on the following targets: `config-c_mb_ieee_3232.lnt, config-c_ml_ieee_3232.lnt, config-c_ms_ieee_3232.lnt, config-hcs12x_c_mb_ieee_3232.lnt, confighcs12x_c_mb_ieee_3232_no_jsr2bsr.lnt, config-hcs12x_c_ml_ieee_3232.lnt, config-hcs12x_c_ml_ieee_3232_const_near.lnt, config-hcs12x_c_ml_ieee_3232_no_jsr2bsr.lnt,config-hcs12x_c_ml_ieee_3232_non_const_near.lnt, config-hcs12x_c_ms_ieee_3232.lnt, config-hcs12x_c_ms_ieee_3232_no_jsr2bsr.lnt`
4. Valid on the following targets: `config-c_mb_ieee_3232.lnt, config-c_mb_ieee_3264.lnt, config-c_ml_ieee_3232.lnt, config-c_ml_ieee_3264.lnt, configc_ms_ieee_3232.lnt, config-c_ms_ieee_3264.lnt, config-hcs12x_c_mb_ieee_3232.lnt, config-hcs12x_c_mb_ieee_3232_no_jsr2bsr.lnt, config-hcs12x_c_mb_ieee_3264.lnt,config-hcs12x_c_mb_ieee_3264_no_jsr2bsr.lnt, config-hcs12x_c_ml_ieee_3232.lnt, config-hcs12x_c_ml_ieee_3232_const_near.lnt, confighcs12x_c_ml_ieee_3232_no_jsr2bsr.lnt, config-hcs12x_c_ml_ieee_3232_non_const_near.lnt, config-hcs12x_c_ml_ieee_3264.lnt, confighcs12x_c_ml_ieee_3264_const_near.lnt, config-hcs12x_c_ml_ieee_3264_no_jsr2bsr.lnt, config-`

hcs12x_c_ml_ieee_3264_non_const_near.lnt, confighcs12x_c_ms_ieee_3232.lnt, config-hcs12x_c_ms_ieee_3232_no_jsr2bsr.lnt, config-hcs12x_c_ms_ieee_3264.lnt, config-hcs12x_c_ms_ieee_3264_no_jsr2bsr.lnt

5. Valid on all targets except: config-c_mb_startup.lnt, config-c_ml_startup.lnt, config-c_ms_startup.lnt, config-hcs12x_c_mb_startup.lnt, confighcs12x_c_mb_startup_no_jsr2bsr.lnt, config-hcs12x_c_ml_ieee_3232_const_near.lnt, config-hcs12x_c_ml_ieee_3232_non_const_near.lnt, confighcs12x_c_ml_ieee_3264_const_near.lnt, config-hcs12x_c_ml_ieee_3264_non_const_near.lnt, config-hcs12x_c_ml_no_float_const_near.lnt, confighcs12x_c_ml_no_float_non_const_near.lnt, config-hcs12x_c_ml_startup.lnt, config-hcs12x_c_ml_startup_const_near.lnt, config-hcs12x_c_ml_startup_no_jsr2bsr.lnt,config-hcs12x_c_ml_startup_non_const_near.lnt, config-hcs12x_c_ms_startup.lnt, config-hcs12x_c_ms_startup_no_jsr2bsr.lnt
6. Valid on the following targets: config-c_mb_ieee_3232.lnt, config-c_mb_no_float.lnt, config-c_ml_ieee_3232.lnt, config-c_ml_no_float.lnt, config-c_ms_ieee_3232.lnt,config-c_ms_no_float.lnt, config-hcs12x_c_mb_ieee_3232.lnt, config-hcs12x_c_mb_ieee_3232_no_jsr2bsr.lnt, config-hcs12x_c_mb_no_float.lnt, confighcs12x_c_mb_no_float_no_jsr2bsr.lnt, config-hcs12x_c_ml_ieee_3232.lnt, config-hcs12x_c_ml_ieee_3232_const_near.lnt, config-hcs12x_c_ml_ieee_3232_no_jsr2bsr.lnt, config-hcs12x_c_ml_ieee_3232_non_const_near.lnt, config-hcs12x_c_ml_no_float.lnt, config-hcs12x_c_ml_no_float_const_near.lnt, confighcs12x_c_ml_no_float_no_jsr2bsr.lnt, config-hcs12x_c_ml_no_float_non_const_near.lnt, config-hcs12x_c_ms_ieee_3232.lnt, config-hcs12x_c_ms_ieee_3232_no_jsr2bsr.lnt,config-hcs12x_c_ms_no_float.lnt, config-hcs12x_c_ms_no_float_no_jsr2bsr.lnt
7. Valid on the following targets: config-c_mb_ieee_3232.lnt, config-c_ms_ieee_3232.lnt, config-hcs12x_c_mb_ieee_3232.lnt, confighcs12x_c_mb_ieee_3232_no_jsr2bsr.lnt, config-

`hcs12x_c_ms_ieee_3232.lnt, config-hcs12x_c_ms_ieee_3232_no_jsr2bsr.lnt`

8. Valid on the following targets: `config-c_mb_ieee_3264.lnt, config-c_ml_ieee_3264.lnt, config-c_ms_ieee_3264.lnt, config-hcs12x_c_mb_ieee_3264.lnt, confighcs12x_c_mb_ieee_3264_no_jsr2bsr.lnt, config-hcs12x_c_ml_ieee_3264.lnt, config-hcs12x_c_ml_ieee_3264_const_near.lnt, config-hcs12x_c_ml_ieee_3264_no_jsr2bsr.lnt, config-hcs12x_c_ml_ieee_3264_non_const_near.lnt, config-hcs12x_c_ms_ieee_3264.lnt, config-hcs12x_c_ms_ieee_3264_no_jsr2bsr.lnt`
9. Valid on the following targets: `config-c_mb_startup.lnt, config-c_ml_startup.lnt, config-c_ms_startup.lnt, config-hcs12x_c_mb_startup.lnt, confighcs12x_c_mb_startup_no_jsr2bsr.lnt, config-hcs12x_c_ml_startup.lnt, config-hcs12x_c_ml_startup_const_near.lnt, config-hcs12x_c_ml_startup_no_jsr2bsr.lnt, confighcs12x_c_ml_startup_non_const_near.lnt, config-hcs12x_c_ms_startup.lnt, config-hcs12x_c_ms_startup_no_jsr2bsr.lnt`
10. Valid on the following targets: `config-c_mb_startup.lnt, config-c_ms_startup.lnt, config-hcs12x_c_mb_startup.lnt, config-hcs12x_c_mb_startup_no_jsr2bsr.lnt, config-hcs12x_c_ml_startup.lnt, config-hcs12x_c_ml_startup_const_near.lnt, config-hcs12x_c_ml_startup_no_jsr2bsr.lnt, config-hcs12x_c_ml_startup_non_const_near.lnt,config-hcs12x_c_ms_startup.lnt, config-hcs12x_c_ms_startup_no_jsr2bsr.lnt`
11. Valid on the following targets: `config-c_mb_startup.lnt, config-c_ml_startup.lnt, config-c_ms_startup.lnt, config-hcs12x_c_mb_startup.lnt, confighcs12x_c_mb_startup_no_jsr2bsr.lnt, config-hcs12x_c_ms_startup.lnt, config-hcs12x_c_ms_startup_no_jsr2bsr.lnt`
12. Valid on the following targets: `config-c_ml_ieee_3232.lnt, config-c_ml_ieee_3264.lnt, config-c_ml_no_float.lnt, config-hcs12x_c_ml_ieee_3232.lnt, confighcs12x_c_ml_ieee_3232_const_near.lnt, config-hcs12x_c_ml_ieee_3232_no_jsr2bsr.lnt, config-`

`hcs12x_c_ml_ieee_3232_non_const_near.lnt, confighcs12x_c_ml_ieee_3264.lnt, config-hcs12x_c_ml_ieee_3264_const_near.lnt, config-hcs12x_c_ml_ieee_3264_no_jsr2bsr.lnt, confighcs12x_c_ml_ieee_3264_non_const_near.lnt, config-hcs12x_c_ml_no_float.lnt, config-hcs12x_c_ml_no_float_const_near.lnt, config-hcs12x_c_ml_no_float_no_jsr2bsr.lnt,config-hcs12x_c_ml_no_float_non_const_near.lnt`

13. Valid on the following targets: `config-c_ml_ieee_3232.lnt, config-hcs12x_c_ml_ieee_3232.lnt, config-hcs12x_c_ml_ieee_3232_const_near.lnt, confighcs12x_c_ml_ieee_3232_no_jsr2bsr.lnt, config-hcs12x_c_ml_ieee_3232_non_const_near.lnt`
14. Valid on the following targets: `config-c_ml_startup.lnt`
15. Valid on the following targets: `config-hcs12x_c_ml_startup.lnt, config-hcs12x_c_ml_startup_const_near.lnt, config-hcs12x_c_ml_startup_no_jsr2bsr.lnt, confighcs12x_c_ml_startup_non_const_near.lnt`
16. Valid on all targets
17. Valid on the following targets: `config-c_ieee_3232.lnt`
18. Valid on the following targets: `config-c_no_float.lnt`