

C2000™ MISRA-C Policy

Elizabeth Joy

ABSTRACT

This application report outlines the C2000™ MISRA-C policy. The report lists the MISRA-C guidelines adhered to by MISRA-C compliant C2000 software. This policy also details the justifications for approved deviations from the MISRA-C standard.

Contents

1	Introduction	2
2	C2000 MISRA-C Policy	2
2.1	MISRA-C Adhered Guidelines	3
2.2	MISRA-C Partially Checked Guidelines	14
2.3	MISRA-C Blanket Deviations	14
2.4	MISRA-C Case-by-Case Deviations	15
3	References	18

List of Tables

1	Adhered Guidelines	3
2	Partially Checked Guidelines	14
3	Blanket Deviations	14
4	Case-by-Case Deviations	15

Trademarks

C2000 is a trademark of Texas Instruments.
All other trademarks are the property of their respective owners.

1 Introduction

The MISRA-C standard is a set of coding guidelines intended to improve the safety, security, portability, and reliability of software written in the C programming language. MISRA-C advises to refrain from using the undefined and unspecified aspects of C language. This standard can also minimize risks if the developer misunderstands or misuses the language specifications. MISRA-C can also identify potential runtime errors that are not flagged by the compiler. MISRA-C is developed by the Motor Industry Software Reliability Association (MISRA). The standard is adopted for embedded software across automotive, industrial, medical, aerospace, and other safety-critical fields. There are three MISRA-C versions to date: MISRA-C:1998, MISRA-C:2004, and MISRA-C:2012. MISRA-C:2012 supports both C99 and C90 language specifications.

In this application report, MISRA-C refers to the MISRA-C:2012 version. A guideline falls into either of two classifications: rule or directive. This classification is denoted by the letter R or D in the corresponding guideline number. A Rule requires only the source code to verify compliance, while a Directive requires source code and may also require additional inputs, like the design documentation, to verify compliance. A guideline belongs to one of three categories: mandatory, required, or advisory. The mandatory guidelines must always be followed for a project to meet MISRA-C compliance. The required guidelines should be complied with unless the developer documents a formal justification. Lastly, the advisory guidelines are suggestions to be implemented if practical, and deviations must be documented^[1].

MISRA-C checks are typically performed after software development and before unit-level testing, as well as before software releases. For C2000 projects, MISRA-C compliance is verified by performing static analysis using the LDRA Testbed Tool. This tool is included in the Liverpool Data Research Associates (LDRA) tool suite. In addition to providing the Testbed tool, the LDRA tool suite consists of tools for unit-level testing and code coverage. Static analysis is an automated software analysis, without program execution, to ensure that the software conforms to a specified coding standard. The LDRA Testbed tool generates reports that show the LDRA standard numbers and the MISRA-C guidelines associated with noncompliant lines of code. The noncompliant code is rewritten or appropriate justification is provided until the reports from static analysis indicate compliance.

This application report discusses the general C2000 MISRA-C policy and outlines the deviations from the MISRA-C standard that have been deemed appropriate by the C2000 team. Some C2000 software packages have adopted this policy, while other packages follow a project-specific policy derived from this policy. There are also some packages that do not comply with MISRA-C. See the corresponding documentation of the software to determine whether it is MISRA-C compliant and which policy is followed.

2 C2000 MISRA-C Policy

The policy is divided into four sections:

- [Adhered Guidelines](#)
- [Partially Checked Guidelines](#)
- [Blanket Deviations](#)
- [Case-by-Case Deviations](#)

The [Adhered Guidelines](#) are the guidelines complied by the software. The [Partially Checked Guidelines](#), [Blanket Deviations](#), and [Case-by-Case Deviations](#) sections provide justification for deviations.

2.1 MISRA-C Adhered Guidelines

Table 1 lists the guidelines of MISRA-C that are adhered to by C2000 projects. If the reports from static analysis indicate violations to any of these guidelines, then the code is rewritten and static analysis is repeated until the violations are removed. As previously mentioned, mandatory guidelines cannot be violated in MISRA-C compliant software. The list of adhered guidelines includes all the guidelines categorized as mandatory by MISRA-C, as well as the majority of required and advisory guidelines.

Table 1. Adhered Guidelines

MISRA-C Guideline [1]	Category [1]	Guideline Description[1]	LDRA Standard(s)[2]	LDRA Standard Description[2]
D.2.1	Required	All source files shall compile without any compilation errors.	None	No Tool Support. C28x compiler is used to compile C2000 projects and ensure there are no errors at compile-time.
D.3.1	Required	All code shall be traceable to documented requirements.	None	No Tool Support. Design review is conducted to be compliant.
D.4.2	Advisory	All usage of assembly language should be documented.	17 S	Code insert found.
D.4.3	Required	Assembly language shall be encapsulated and isolated.	88 S	Procedure is not pure assembler.
D.4.4	Advisory	Sections of code should not be 'commented out'.	302 S	Comment possibly contains code.
D.4.5	Advisory	Identifiers in the same namespace with overlapping visibility should be typographically unambiguous.	217 S 67 X	Names only differ by case. Identifier is typographically ambiguous.
D.4.6	Advisory	typedefs that indicate size and signedness should be used in place of the basic numerical types.	90 S 495 S	Basic type declaration used. Typedef name has no size indication.
D.4.7	Required	If a function returns error information, then that error information shall be tested.	91 D	Function return value potentially unused.
D.4.8	Advisory	If a pointer to a structure or union is never dereferenced within a translation unit, then the implementation of the object should be hidden.	104 D	Structure implementation not hidden.
D.4.10	Required	Precautions shall be taken in order to prevent the contents of a header file being included more than once.	243 S	Included file not protected with #define.
D.4.11	Required	The validity of values passed to library functions shall be checked.	None	No Tool Support. Code review is conducted to be compliant.
D.4.12	Required	Dynamic memory allocation shall not be used.	44 S	Use of banned function or variable.
D.4.13	Advisory	Functions which are designed to provide operations on a resource should be called in an appropriate sequence.	None	No Tool Support. Code review is conducted to be compliant.
R.1.1	Required	The program shall contain no violations of the standard C syntax and constraints, and shall not exceed the translation limits of the implementation.	21 S	Number of parameters does not match.
			145 S	#if has invalid expression.
			323 S	Switch has more than one default case.
			345 S	Bit operator with floating point operand.
			404 S	Array initialisation has too many items.
			481 S	Array with no bounds in struct.
			580 S	Macro redefinition without using #undef.
			615 S	Conditional operator has incompatible types.

Table 1. Adhered Guidelines (continued)

MISRA-C Guideline [1]	Category [1]	Guideline Description[1]	LDRA Standard(s)[2]	LDRA Standard Description[2]
R.1.3	Required	There shall be no occurrence of undefined or critical unspecified behaviour.	82 D	fsetpos values not generated by fgetpos.
			83 D	Potentially repeated call to ungetc.
			84 D	No fseek or flush before I/O.
			87 D	Illegal shared object in signal handler.
			89 D	Illegal use of raise in signal handler.
			5 Q	File does not end with new line.
			21 S	Number of parameters does not match.
			44 S	Use of banned function or variable.
			64 S	Void procedure used in expression.
			65 S	Void variable passed as parameter.
			113 S	Non standard character in source.
			118 S	main must be int (void) or int (int,char*[]).
			176 S	Non standard escape sequence in source.
			296 S	Function declared at block scope.
			324 S	Macro call has wrong number of parameters.
			335 S	Operator defined contains illegal items.
			336 S	#if expansion contains define operator.
			412 S	Undefined behaviour, \ before E-O-F.
			450 S	Wide string and string concatenated.
			465 S	Struct/union not completely specified.
			482 S	Incomplete structure referenced.
			486 S	Incorrect number of formats in output function.
			487 S	Insufficient space allocated.
			489 S	Insufficient space for operation.
			497 S	Type is incomplete in translation unit.
			573 S	Macro concatenation of uni char names.
			576 S	Function pointer is of wrong type.
			582 S	const object reassigned.
			587 S	Const local variable not immediately initialised.
			589 S	Format is not appropriate type.
			590 S	Mode fault in fopen.
			608 S	Use of explicitly undefined language feature.
			66 X	Insufficient array space at call.
			70 X	Array has insufficient space.
			71 X	Insufficient space for copy.
R.2.3	Advisory	A project should not contain unused type declarations.	413 S	User type declared but not used in code analysed.
R.2.4	Advisory	A project should not contain unused tag declarations.	413 S	User type declared but not used in code analysed.
R.2.6	Advisory	A function should not contain unused label declarations.	610 S	Label is unused.
R.2.7	Advisory	There should be no unused parameters in functions.	1 D	Unused procedure parameter.
			15 D	Unused procedural parameter.
R.3.1	Required	The character sequences /* and // shall not be used within a comment.	119 S	Nested comment found.

Table 1. Adhered Guidelines (continued)

MISRA-C Guideline [1]	Category [1]	Guideline Description[1]	LDRA Standard(s)[2]	LDRA Standard Description[2]
R.3.2	Required	Line-splicing shall not be used in // comments.	611 S	Line splice used in // comment.
R.4.1	Required	Octal and hexadecimal escape sequences shall be terminated.	176 S	Non standard escape sequence in source.
R.4.2	Advisory	Trigraphs should not be used.	81 S	Use of trigraph.
R.5.6	Required	A typedef name shall be a unique identifier.	112 S	Typedef name redeclared.
			374 S	Name conflict with typedef.
			11 X	Identifier reuse: tag vs typedef.
			16 X	Identifier reuse: typedef vs variable.
			17 X	Identifier reuse: typedef vs label (MR).
			18 X	Identifier reuse: typedef vs typedef.
			19 X	Identifier reuse: typedef vs procedure parameter.
			20 X	Identifier reuse: persistent var vs typedef.
			22 X	Identifier reuse: typedef vs component.
			23 X	Identifier reuse: typedef vs enum constant.
R.5.7	Required	A tag name shall be a unique identifier.	24 X	Identifier reuse: typedef vs procedure.
			325 S	Inconsistent use of tag.
			4 X	Identifier reuse: struct/union tag repeated.
			5 X	Identifier reuse: struct vs union.
			6 X	Identifier reuse: struct/union tag vs enum tag.
			7 X	Identifier reuse: tag vs procedure.
			8 X	Identifier reuse: tag vs procedure parameter.
			9 X	Identifier reuse: tag vs variable.
			10 X	Identifier reuse: tag vs label (MR).
			11 X	Identifier reuse: tag vs typedef.
			13 X	Identifier reuse: tag vs component.
			14 X	Identifier reuse: tag vs enum constant.
			15 X	Identifier reuse: persistent var vs tag.

Table 1. Adhered Guidelines (continued)

MISRA-C Guideline [1]	Category [1]	Guideline Description[1]	LDRA Standard(s)[2]	LDRA Standard Description[2]
R.5.8	Required	Identifiers that define objects or functions with external linkage shall be unique.	1 S	Procedure name reused.
			7 X	Identifier reuse: tag vs procedure.
			15 X	Identifier reuse: persistent var vs tag.
			20 X	Identifier reuse: persistent var vs typedef.
			24 X	Identifier reuse: typedef vs procedure.
			25 X	Identifier reuse: procedure vs procedure param.
			26 X	Identifier reuse: persistent var vs label (MR).
			27 X	Identifier reuse: persist var vs persist var.
			28 X	Identifier reuse: persistent var vs var.
			29 X	Identifier reuse: persistent var vs procedure.
			30 X	Identifier reuse: persistent var vs proc param.
			31 X	Identifier reuse: procedure vs procedure.
			32 X	Identifier reuse: procedure vs var.
			33 X	Identifier reuse: procedure vs label (MR).
			35 X	Identifier reuse: proc vs component.
			36 X	Identifier reuse: proc vs enum constant.
			38 X	Identifier reuse: persistent var vs component.
			39 X	Identifier reuse: persistent var vs enum constant.
R.5.9	Advisory	Identifiers that define objects or functions with internal linkage should be unique.	1 S	Procedure name reused.
			7 X	Identifier reuse: tag vs procedure.
			15 X	Identifier reuse: persistent var vs tag.
			20 X	Identifier reuse: persistent var vs typedef.
			24 X	Identifier reuse: typedef vs procedure.
			25 X	Identifier reuse: procedure vs procedure param.
			26 X	Identifier reuse: persistent var vs label (MR).
			27 X	Identifier reuse: persist var vs persist var.
			28 X	Identifier reuse: persistent var vs var.
			29 X	Identifier reuse: persistent var vs procedure.
			30 X	Identifier reuse: persistent var vs proc param.
			31 X	Identifier reuse: procedure vs procedure.
			32 X	Identifier reuse: procedure vs var.
			33 X	Identifier reuse: procedure vs label (MR).
			35 X	Identifier reuse: proc vs component.
			36 X	Identifier reuse: proc vs enum constant.
			38 X	Identifier reuse: persistent var vs component.
			39 X	Identifier reuse: persistent var vs enum constant.
R.6.1	Required	Bit fields shall only be declared with an appropriate type.	73 S	Bit field not signed or unsigned int.
			520 S	Bit field is not bool or explicit integral.
R.6.2	Required	Single-bit named bit fields shall not be of a signed type.	72 S	Signed bit field less than 2 bits wide.
R.7.1	Required	Octal constants shall not be used.	83 S	Octal number found.

Table 1. Adhered Guidelines (continued)

MISRA-C Guideline [1]	Category [1]	Guideline Description[1]	LDRA Standard(s)[2]	LDRA Standard Description[2]
R.7.2	Required	A "u" or "U" suffix shall be applied to all integer constants that are represented in an unsigned type.	331 S	Literal value requires a U suffix.
			550 S	Unsuffix hex or octal is unsigned, add U.
R.7.3	Required	The lowercase character 'l' shall not be used in a literal suffix.	252 S	Lower case suffix to literal number.
R.7.4	Required	A string literal shall not be assigned to an object unless the type of the object is "pointer to const-qualified char"	157 S	Modification of string literal.
			623 S	String assigned to non const object.
R.8.1	Required	Types shall be explicitly specified.	20 S	Parameter not declared explicitly.
			135 S	Parameter list is KR.
			326 S	Declaration is missing type.
R.8.2	Required	Function types shall be in prototype form with named parameters.	37 S	Procedure parameter has a type but no identifier.
			63 S	Empty parameter list to procedure/function.
			135 S	Parameter list is KR.
R.8.3	Required	All declarations of an object or function shall use the same names and type qualifiers.	36 D	Prototype and definition name mismatch.
			63 X	Function prototype/defn param type mismatch (MR).
R.8.4	Required	A compatible declaration shall be visible when an object or function with external linkage is defined.	36 D	Prototype and definition name mismatch.
			106 D	No prototype for non-static function.
			102 S	Function and prototype return inconsistent (MR).
			103 S	Function and prototype param inconsistent (MR).
			1 X	Declaration types do not match across a system.
			62 X	Function prototype/defn return type mismatch (MR).
R.8.5	Required	An external object or function shall be declared once in one and only one file.	63 X	Function prototype/defn param type mismatch (MR).
			60 D	External object should be declared only once.
			110 D	More than one prototype for same function.
R.8.6	Required	An identifier with external linkage shall have exactly one external definition.	172 S	Variable declared multiply.
			26 D	External object should be declared only once.
			33 D	More than one prototype for same function.
			34 D	Variable declared multiply.
R.8.7	Advisory	Functions and objects should not be defined with external linkage if they are referenced in only one translation unit.	63 D	
R.8.8	Required	The static storage class specifier shall be used in all declarations of objects and functions that have internal linkage.	27 D	Variable should be declared static.
			61 D	Procedure should be declared static.
			461 S	Identifier with ambiguous linkage.
			553 S	Function and proto should both be static.
R.8.10	Required	An inline function shall be declared with the static storage class.	575 S	Linkage differs from previous declaration.
R.8.11	Required	When an array with external linkage is declared, its size should be explicitly specified.	612 S	Inline function should be declared static.
			127 S	Array has no bounds specified.

Table 1. Adhered Guidelines (continued)

MISRA-C Guideline [1]	Category [1]	Guideline Description[1]	LDRA Standard(s)[2]	LDRA Standard Description[2]
R.8.12	Required	Within an enumerator list, the value of an implicitly-specified enumeration constant shall be unique.	630 S	Duplicated enumeration value.
R.8.13	Advisory	A pointer should point to a const-qualified type whenever possible.	120 D	Pointer parameter should be declared const.
R.8.14	Required	The restrict type qualifier shall not be used.	613 S	Use of restrict keyword.
R.9.1	Mandatory	The value of an object with automatic storage duration shall not be read before it has been set.	53 D	Attempt to use uninitialised pointer.
			69 D	UR anomaly, variable used before assignment.
			631 S	Declaration not reachable.
R.9.2	Required	The initializer for an aggregate or union shall be enclosed in braces.	105 S	Initialisation brace { } fault.
			627 S	Initialiser both positional and designational.
R.9.3	Required	Arrays shall not be partially initialized.	397 S	Array initialisation has insufficient items.
			627 S	Initialiser both positional and designational.
R.9.4	Required	An element of an object shall not be initialised more than once.	620 S	Initialisation designator duplicated.
R.9.5	Required	Where designated initialisers are used to initialize an array object the size of the array shall be specified explicitly.	127 S	Array has no bounds specified.
R.10.2	Required	Expressions of essentially character type shall not be used inappropriately in addition and subtraction operations.	96 S	Use of mixed mode arithmetic.
			329 S	Operation not appropriate to plain char.
R.10.5	Advisory	The value of an expression should not be cast to an inappropriate essential type.	93 S	Value is not of appropriate type.
R.10.6	Required	The value of a composite expression shall not be assigned to an object with wider essential type.	451 S	No cast for widening complex float expression.
			452 S	No cast for widening complex int expression.
R.10.7	Required	If a composite expression is used as one operand of an operator in which the usual arithmetic conversions are performed then the other operand shall not have wider essential type.	451 S	No cast for widening complex float expression.
			452 S	No cast for widening complex int expression.
R.10.8	Required	The value of a composite expression shall not be cast to a different essential type category or a wider essential type.	332 S	Widening cast on complex integer expression.
			333 S	Widening cast on complex float expression.
			441 S	Float cast to non-float.
			442 S	Signed integral type cast to unsigned.
			443 S	Unsigned integral type cast to signed.
			444 S	Integral type cast to non-integral.
R.11.2	Required	Conversions shall not be performed between a pointer to incomplete and any other type.	94 S	Casting operation on a pointer.
			95 S	Casting operation to a pointer.
			439 S	Cast from pointer to integral type.
			440 S	Cast from integral type to pointer.
			554 S	Cast to an unrelated type.
R.11.5	Advisory	A conversion should not be performed from pointer to void into pointer to object.	95 S	Casting operation to a pointer.
R.11.7	Required	A cast shall not be performed between pointer to object and a non-integer arithmetic type.	94 S	Casting operation on a pointer.
			95 S	Casting operation to a pointer.
			439 S	Cast from pointer to integral type.
			440 S	Cast from integral type to pointer.

Table 1. Adhered Guidelines (continued)

MISRA-C Guideline [1]	Category [1]	Guideline Description[1]	LDRA Standard(s)[2]	LDRA Standard Description[2]
R.11.9	Required	The macro NULL shall be the only permitted form of integer null pointer constant.	531 S	Literal zero used in pointer context.
R.12.1	Advisory	The precedence of operators within expressions should be made explicit.	49 S	Logical conjunctions need brackets.
			361 S	Expression needs brackets.
R.12.2	Required	The right hand operand of a shift operator shall lie in the range zero to one less than the width in bits of the essential type of the left hand operand.	51 S	Shifting value too far.
			403 S	Negative (or potentially negative) shift.
R.12.3	Advisory	The comma operator should not be used.	53 S	Use of comma operator.
R.12.4	Advisory	Evaluation of constant expressions should not lead to unsigned integer wrap-around.	493 S	Numeric overflow.
			494 S	Numeric underflow.
R.13.1	Required	Initialiser lists shall not contain persistent side effects.	35 D	Expression has side effects.
			1 Q	Call has execution order dependent side effects.
			9 S	Assignment operation in expression.
			30 S	Deprecated usage of ++ or -- operators found.
			132 S	Assignment operator in boolean expression.
			134 S	Volatile variable in complex expression.
R.13.2	Required	The value of an expression and its persistent side effects shall be the same under all permitted evaluation orders.	35 D	Expression has side effects.
			72 D	Potential side effect problem in expression.
			74 D	Potential side effect from repeated function call.
			1 Q	Call has execution order dependent side effects.
			9 S	Assignment operation in expression.
			30 S	Deprecated usage of ++ or -- operators found.
			134 S	Volatile variable in complex expression.
R.13.3	Advisory	A full expression containing an increment (++) or decrement (--) operator should have no other potential side effects other than that caused by the increment or decrement operator.	30 S	Deprecated usage of ++ or -- operators found.
R.13.4	Advisory	The result of an assignment operator should not be used.	9 S	Assignment operation in expression.
			132 S	Assignment operator in boolean expression.
R.13.5	Required	The right hand operand of a logical && or operator shall not contain persistent side effects.	35 D	Expression has side effects.
			1 Q	Call has execution order dependent side effects.
			406 S	Use of ++ or -- on RHS of && or operator.
			408 S	Volatile variable accessed on RHS of && or .
R.13.6	Mandatory	The operand of the sizeof operator shall not contain any expression which has potential side effects.	54 S	Sizeof operator with side effects.
R.14.1	Required	A loop counter shall not have essentially floating type.	39 S	Unsuitable type for loop variable.

Table 1. Adhered Guidelines (continued)

MISRA-C Guideline [1]	Category [1]	Guideline Description[1]	LDRA Standard(s)[2]	LDRA Standard Description[2]
R.14.2	Required	A for loop shall be well-formed.	55 D	Modification of loop counter in loop body.
			270 S	For loop initialisation is not simple.
			271 S	For loop incrementation is not simple.
			429 S	Empty middle expression in for loop.
			430 S	Inconsistent usage of loop control variable.
			581 S	Loop conditions are independent.
R.14.3	Required	Controlling expressions shall not be invariant.	139 S	Construct leads to infeasible code.
			140 S	Infeasible loop condition found.
R.14.4	Required	The controlling expression of an if statement and the controlling expression of an iteration-statement shall have essentially Boolean type.	114 S	Expression is not Boolean.
R.15.1	Advisory	The goto statement should not be used.	13 S	goto detected.
R.15.2	Required	The goto statement shall jump to a label declared later in the same function.	509 S	goto label is backwards.
R.15.3	Required	Any label referenced by a goto statement shall be declared in the same block, or in any block enclosing the goto statement.	511 S	Jump into nested block.
R.15.4	Advisory	There should be no more than one break or goto statement used to terminate any iteration statement.	409 S	More than one break or goto statement in loop.
R.15.6	Required	The body of an iteration-statement or a selection-statement shall be a compound statement.	11 S	No brackets to loop body (added by Testbed).
			12 S	No brackets to then/else (added by Testbed).
			428 S	No {} for switch (added by Testbed).
R.15.7	Required	All if . . else if constructs shall be terminated with an else statement.	59 S	Else alternative missing in if.
			477 S	Empty else clause following else if.
R.16.1	Required	All switch statements shall be well-formed.	385 S	MISRA switch statement syntax violation.
R.16.2	Required	A switch label shall only be used when the most closely-enclosing compound statement is the body of a switch statement.	245 S	Case statement in nested block.
R.16.3	Required	An unconditional break statement shall terminate every switch-clause.	62 S	Switch case not terminated with break.
R.16.4	Required	Every switch statement shall have a default label.	48 S	No default case in switch statement.
			410 S	Switch empty default has no comment.
R.16.5	Required	A default label shall appear as either the first or the last switch label of a switch statement.	322 S	Default is not last case of switch.
R.16.6	Required	Every switch statement shall have at least two switch-clauses.	60 S	Empty switch statement.
			61 S	Switch contains default only.
R.16.7	Required	A switch-expression shall not have essentially Boolean type.	121 S	Use of boolean expression in switch.
R.17.1	Required	The features of <stdarg.h> shall not be used.	44 S	Use of banned function or variable.
R.17.2	Required	Functions shall not call themselves, either directly or indirectly.	6 D	Recursion in procedure calls found.
			1 U	Inter-file recursion found.
R.17.3	Mandatory	A function shall not be declared implicitly.	496 S	Function call with no prior declaration.

Table 1. Adhered Guidelines (continued)

MISRA-C Guideline [1]	Category [1]	Guideline Description[1]	LDRA Standard(s)[2]	LDRA Standard Description[2]
R.17.4	Mandatory	All exit paths from a function with non-void return type shall have an explicit return statement with an expression.	2 D	Function does not return a value on all paths.
			36 S	Function has no return statement.
			66 S	Function with empty return expression.
R.17.5	Advisory	The function argument corresponding to a parameter declared to have an array type shall have an appropriate number of elements.	64 X	Array bound exceeded at call.
R.17.6	Mandatory	The declaration of an array parameter shall not contain the static keyword between the [].	614 S	Use of static keyword in array parameter.
R.17.8	Advisory	A function parameter should not be modified.	14 D	Attempt to change parameter passed by value.
			149 S	Reference parameter to procedure is reassigned.
R.18.1	Required	A pointer resulting from arithmetic on a pointer operand shall address an element of the same array as that pointer operand.	47 S	Array bound exceeded.
			436 S	Declaration does not specify an array.
			567 S	Pointer arithmetic is not on array.
			64 X	Array bound exceeded at call.
			68 X	Parameter indexing array too big at call.
			69 X	Global array bound exceeded at use.
R.18.2	Required	Subtraction between pointers shall only be applied to pointers that address elements of the same array.	72 X	Parameter indexing array too small at call.
			438 S	Pointer subtraction not addressing one array.
R.18.3	Required	The relational operators >, >=, < and <= shall not be applied to objects of pointer type except where they point into the same object.	437 S	< > <= >= used on different object pointers.
R.18.5	Advisory	Declarations should contain no more than two levels of pointer nesting.	80 S	Pointer indirection exceeds 2 levels.
R.18.6	Required	The address of an object with automatic storage shall not be copied to another object that persists after the first object has ceased to exist.	42 D	Local pointer returned in function result.
			77 D	Local structure returned in function result.
			71 S	Pointer assignment to wider scope.
			565 S	Assignment to wider scope.
R.18.7	Required	Flexible array members shall not be declared.	481 S	Array with no bounds in struct.
R.18.8	Required	Variable-length array types shall not be used.	621 S	Variable-length array declared.
R.19.1	Mandatory	An object shall not be assigned or copied to an overlapping object.	480 S	String function params access same variable.
			545 S	Assignment of overlapping storage.
R.20.1	Advisory	#include directives should only be preceded by preprocessor directives or comments.	75 S	Executable code before an included file.
			338 S	#include preceded by non preproc directives.
R.20.2	Required	The ', " or \ characters and the /* or // character sequences shall not occur in a header file name.	100 S	#include filename is non conformant.
R.20.3	Required	The #include directive shall be followed by either a <filename> or "filename" sequence.	427 S	Filename in #include not in < > or " ".

Table 1. Adhered Guidelines (continued)

MISRA-C Guideline [1]	Category [1]	Guideline Description[1]	LDRA Standard(s)[2]	LDRA Standard Description[2]
R.20.4	Required	A macro shall not be defined with the same name as a keyword.	86 S	Attempt to define reserved word.
			580 S	Macro redefinition without using #undef.
			626 S	#define of keyword.
R.20.5	Advisory	#undef should not be used.	68 S	#undef used.
			426 S	#undef used in a block.
R.20.6	Required	Tokens that look like a preprocessing directive shall not occur within a macro argument.	341 S	Preprocessor construct as macro parameter.
R.20.7	Required	Expressions resulting from the expansion of macro parameters shall be enclosed in parentheses.	78 S	Macro parameter not in brackets.
			361 S	Expression needs brackets.
R.20.8	Required	The controlling expression of a #if or #elif preprocessing directive shall evaluate to 0 or 1.	616 S	Preprocessor result not 0 or 1.
R.20.9	Required	All identifiers used in the controlling expression of #if or #elif preprocessing directives shall be #define'd before evaluation.	337 S	Undefined macro variable in #if.
R.20.11	Required	A macro parameter immediately following a # operator shall not immediately be followed by a ## operator	76 S	More than one of # or ## in a macro.
R.20.12	Required	A macro parameter used as an operand to the # or ## operators, which is itself subject to further macro replacement, shall only be used as an operand to these operators.	125 S	Use of ## or # in a macro.
R.20.13	Required	A line whose first token is # shall be a valid preprocessing directive.	147 S	Spurious characters after preprocessor directive.
			342 S	Extra chars after preprocessor directive.
R.20.14	Required	All #else, #elif and #endif preprocessor directives shall reside in the same file as the #if, #ifdef or #ifndef directive to which they are related.	126 S	A #if has no #endif in the same file.
			343 S	#else has no #if, etc in the same file.
R.21.1	Required	#define and #undef shall not be used on a reserved identifier or reserved macro name.	86 S	Attempt to define reserved word.
			156 S	Use of 'defined' keyword in macro body.
			219 S	User name starts with underscore.
R.21.3	Required	The memory allocation and deallocation functions of <stdlib.h> shall not be used.	44 S	Use of banned function or variable.
R.21.4	Required	The standard header file <setjmp.h> shall not be used.	43 S	Use of setjmp/longjmp.
R.21.5	Required	The standard header file <signal.h> shall not be used.	130 S	Included file is not permitted.
R.21.6	Required	The Standard Library input/output routines shall not be used.	44 S	Use of banned function or variable.
			130 S	Included file is not permitted.
R.21.7	Required	The atof, atoi, atol and atoll functions of <stdlib.h> shall not be used.	44 S	Use of banned function or variable.
R.21.8	Required	The library functions abort, exit, getenv and system of <stdlib.h> shall not be used.	44 S	Use of banned function or variable.
			122 S	Use of abort, exit, etc.
R.21.9	Required	The library functions bsearch and qsort of <stdlib.h> shall not be used.	44 S	Use of banned function or variable.
R.21.10	Required	The Standard Library time and date routines shall not be used.	44 S	Use of banned function or variable.
			130 S	Included file is not permitted.

Table 1. Adhered Guidelines (continued)

MISRA-C Guideline [1]	Category [1]	Guideline Description[1]	LDRA Standard(s)[2]	LDRA Standard Description[2]
R.21.11	Required	The standard header file <tgmath.h> shall not be used.	130 S	Included file is not permitted.
R.21.12	Advisory	The exception handling features of <fenv.h> should not be used.	44 S	Use of banned function or variable.
R.22.1	Required	All resources obtained dynamically by means of Standard Library functions shall be explicitly released.	49 D	File pointer not closed on exit.
			50 D	Memory not freed after last reference.
			75 D	Attempt to open file pointer more than once.
R.22.2	Mandatory	A block of memory shall only be freed if it was allocated by means of a Standard Library function.	51 D	Attempt to read from freed memory.
			407 S	free used on string.
			483 S	free parameter is not heap item.
			483 S	Attempt to use already freed object.
R.22.3	Required	The same file shall not be open for read and write access at the same time on different streams.	103 D	File opened both read and write.
R.22.4	Mandatory	There shall be no attempt to write to a stream which has been opened as read-only.	98 D	Attempt to write to file opened read only.
R.22.5	Mandatory	A pointer to a FILE object shall not be dereferenced.	591 S	Inappropriate use of file pointer.
R.22.6	Mandatory	The value of a pointer to a FILE shall not be used after the associated stream has been closed.	48 D	Attempt to write to unopened file.
R.22.5	Mandatory	A pointer to a FILE object shall not be dereferenced.	591 S	Inappropriate use of file pointer.
R.22.6	Mandatory	The value of a pointer to a FILE shall not be used after the associated stream has been closed.	48 D	Attempt to write to unopened file.

2.2 MISRA-C Partially Checked Guidelines

Table 2 lists the guidelines of MISRA-C that cannot be completely verified through static analysis. Due to tool limitations, the source code is only partially checked for compliance to these guidelines.

Table 2. Partially Checked Guidelines

MISRA-C Guideline [1]	Category [1]	Guideline Description[1]	LDRA Standard(s)[2]	Reasoning
D.4.1	Required	Run-time failures shall be minimised	45 D - Pointer not checked for null before use	MISRA guideline D.4.1 is partially implemented by the tool. Dynamic analysis can further reduce run-time errors. 45 D is waived for global pointers and linker defined pointers, where pointer NULL check is performed once in the beginning of the function instead of multiple checks throughout.
R.12.4	Advisory	Evaluation of constant expressions should not lead to unsigned integer wrap-around	493 S - Numeric overflow	493 S is only waived when the expression has been carefully reviewed and it is confirmed that there is no chance of wrap around. For example, regIndex is always greater than or equal to 1 here, which the Testbed tool does not consider. <pre>regVal = HWRREG(baseAddr + (0x100 + (((regIndex) - 1U) * 0x20U)));</pre>

2.3 MISRA-C Blanket Deviations

Table 3 lists the guidelines of MISRA-C that are always waived. The source code is not checked for compliance to these guidelines. The static analysis tool is configured such that violations to these guidelines are not reported. All of these guidelines are categorized as advisory by the MISRA-C standard.

Table 3. Blanket Deviations

MISRA-C Guideline [1]	Category [1]	Guideline Description[1]	LDRA Standard(s)[2]	Reasoning
D.4.9	Advisory	Use of function like macro, The #define pre-processor directive shall not be used to create inline macros.	340 S (off) - Use of function like macro	340 S is always waived as small function-like macros are used to tradeoff between the speed and readability of the firmware. This guideline is disabled in the tool.
R.1.2	Advisory	Language extensions should not be used.	110 S (off) - Use of single line comment 143 S (off) - Curly brackets used 632 S (off) - Use of // comment in pre-processor directive or macro defn	110 S, 143 S, 632 S are always waived because they are comment-related advisory rules, and they conflict with the Texas Instruments coding standard for comments.
R.2.5	Advisory	A project should not contain unused macro declarations.	628 S (off)	628 S is always waived as there may be macros unused by Driverlib source but used by application source. These violations can appear when static analysis is run on Driverlib as opposed to the application code. This situation is justifiable as the macros do not affect functionality.

2.4 MISRA-C Case-by-Case Deviations

Table 4 lists the rules of MISRA-C that are followed when reasonably practical, but can be waived in approved situations. Each instance of a violation from these guidelines is reviewed by the software team. The developer determines whether the reasoning documented in this policy applies to the violation. If the deviation is approved by this policy, an inspected comment is placed directly above the violating line of code to suppress the violation. In the LDRA Testbed Tool, the following format is used for these comments:

```
/* LDRA_INSPECTED <LDRA Standard Number> MR:< [MISRA-C Guideline Number(s)]> Text justifying violation */
```

For example, the following comment indicates that the LDRA standard 69 S maps to MISRA Directive 1.1, and that this violation is waived for the reasoning given in Table 4.

```
/*LDRA_INSPECTED 69 S MR:D.1.1 Comment_69S*/
```

Table 4. Case-by-Case Deviations

MISRA-C Guideline [1]	Category[1]	Guideline Description[1]	LDRA Standard(s)[2]	Reasoning
D.1.1	Required	Any implementation-defined behaviour on which the output of the program depends shall be documented and understood.	69 S - #pragma used	69 S is only waived for the following pragmas: DATA_SECTION, SET_CODE_SECTION, DATA_ALIGN and CODE_STATE. They are used to allocate code and data memory sections to certain RAM area. Use of pragmas other than these is not allowed.
R.2.1	Required	A project should not contain unreachable code.	35 S (off) - Static procedure is not explicitly called in code analysed.	Blanket - 35 S is always waived as static inline functions from Driverlib header files will not be called in the module. This is acceptable as the functions would be called in the application.
			28 D - Potentially infinite loop found	Case-by-Case - 28 D is only waived when there is a deliberate need to have an infinite loop inside an illegal operation handler for debugging purposes. Code must be reviewed to remove any unintended case of infinite loop.
R.2.2	Required	There shall be no dead code.	65 D (off) - Void function has no side effects.	Blanket - 65 D is always waived as there are Driverlib APIs with void return types when a return value is unnecessary. This is done to reduce size and increase speed.
			8 D - DD data flow anomalies found.	Case-by-Case - 8 D and 105 D are only waived in specific situations. One such situation is the need to do dummy reads of some registers. Another justifiable situation is redefining a variable without being referenced after its definition.
			105 D - DU anomaly dead code, variable value is unused on all paths.	
R.5.1	Required	External identifiers shall be distinct.	17 D - Identifier not unique within *** characters.	17 D and 61 X are only waived when more than 31 characters are necessary to give an external identifier a meaningful name.
			61 X - Identifier match in *** chars.	
R.5.2	Required	Identifiers declared in the same scope and name space shall be distinct.	17 D - Identifier not unique within *** characters.	See R.5.1 reasoning for 17 D and 61 X.
			61 X - Identifier match in *** chars.	

Table 4. Case-by-Case Deviations (continued)

MISRA-C Guideline [1]	Category[1]	Guideline Description[1]	LDRA Standard(s)[2]	Reasoning
R.5.3	Required	An identifier declared in an inner scope shall not hide an identifier declared in an outer scope.	17 D - Identifier not unique within *** characters. 61 X - Identifier match in *** chars.	See R.5.1 reasoning for 17 D and 61 X.
R.5.4	Required	Macro identifiers shall be distinct.	384 S - Identifier matches macro name in 31 chars. 61 X - Identifier match in *** chars.	See R.5.1 reasoning for 17 D and 61 X.
R.5.5	Required	Identifiers shall be distinct from macro names.	384 S - Identifier matches macro name in 31 chars.	See R.5.1 reasoning for 17 D and 61 X.
R.8.9	Advisory	An object should be defined at block scope if its identifier only appears in a single function.	25 D - Scope of variable could be reduced.	25 D is only waived when necessary to expose a status obtained inside a function as a global to be available for use by other files.
R.10.1	Required	Operands shall not be of an inappropriate essential type.	96 S - Value is not of appropriate type.	96 S is waived when tool generates false positives in shift operations. The shifting amount (right hand operand) type must be always unsigned and it cannot match the type of the shifting variable (left hand operand).
			120 S - Use of bit operator on signed type.	120 S is case by case as bit operation on enums required as it directly maps to hardware register modes.
			123 S - Use of underlying enum representation value.	123 S is waived as the tool checks for this standard more stringently than recommended by MISRA rules. It is acceptable if the enum has been used with type cast.
R.10.3	Required	The value of an expression shall not be assigned to an object with a narrower essential type or of a different essential type category.	101 S - Function return type inconsistent.	101 S, 330 S, 434 S are only waived for compiler intrinsics. C28x compiler has intrinsics to take care of access requests for a byte or 4 bytes. These intrinsics are __byte() and __byte_peripheral_32() which return a reference. LDRA tool does not have a way to support references in C as it is outside of the language spec for its static analysis rules. 488 S is waived as it is a tool issue generating a false positive for reads using the HWREGH() macro.
			330 S - Implicit conversion of underlying type.	
			434 S - Signed/unsigned conversion without cast.	
R.10.4	Required	Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category.	93 S - Value is not of appropriate type.	93 S is waived when minimal typecast is required for assigning address from void * to essential type.
			488 S - Value outside range of underlying type.	488 S is waived as it is a tool issue generating false positive for reads using the HWREGH() macro.
R.11.1	Required	Conversions shall not be performed between a pointer to a function and any other type.	606 S - Cast involving function pointer.	606 S is only waived for casting interrupt service routines when necessary. For example: <code>HWREG(PIEVECTTABLE_BASE + (2U * i)) =(uint32_t)Interrupt_defaultHandler;</code>

Table 4. Case-by-Case Deviations (continued)

MISRA-C Guideline [1]	Category[1]	Guideline Description[1]	LDRA Standard(s)[2]	Reasoning
R.11.3	Required	A cast shall not be performed between a pointer to object type and a pointer to a different object type.	94 S - Casting operation on a pointer.	94 S, 95 S, 554 S are only waived when minimal typecast on pointers is done in software for more effective throughput. In many cases, byte pointer is type casted to word pointer to perform word read/write. These operations should be tested thoroughly in software.
			95 S - Casting operation to a pointer.	
			554 S - Cast to an unrelated type	
R.11.4	Advisory	A conversion should not be performed between a pointer to object and an integer type.	439 S - Cast from pointer to integral type.	439 S is waived for certain situations when it is acceptable to cast from a pointer to integer type. Memory mapped registers and other hardware features access require typecasting of pointer to integer. There are cases when it is necessary to typecast a pointer to an integer for pointer arithmetic.
			440 S (off) - Cast from integral type to pointer.	440 S is always waived as there is a need to cast from integral type to pointer for register accesses.
R.11.6	Required	A cast shall not be performed between pointer to void and an arithmetic type.	439 S - Cast from pointer to integral type.	See R.11.4 reasoning for 439 S and 440 S.
			440 S (off) - Cast from integral type to pointer.	
R.11.8	Required	A cast shall not remove any const or volatile qualification from the type pointed to by a pointer.	203 S - Cast on a constant value.	203 S can be waived if the cast does not lead to undefined behavior. Specifically this occurs when the cast removes the const qualification on a pointer to const type while performing some arithmetic check after reading value.
R.17.7	Required	The value returned by a function having non-void return type shall be used.	382 S - (void) missing for discarded return value.	382 S is only waived for compiler intrinsics <code>__byte()</code> and <code>__byte_peripheral_32()</code> . The intrinsics return references and LDRA does not support references for MISRA-C Static analysis. This reference is not discarded as the violation implied. Rather, the reference is used to write data. More details about these intrinsics can be found in the C2000 Compiler User's Guide .
R.18.1	Required	A pointer resulting from arithmetic on a pointer operand shall address an element of the same array as that pointer operand.	47 S - Array bound exceeded.	47 S is only waived when the LDRA tool does not correctly report the array out-of-bound check.
			436 S - Declaration does not specify an array.	436 S and 567 S are only waived when necessary to access a repeated set of registers. Using pointer arithmetic results in more readable code.
			567 S - Pointer arithmetic is not on array.	
R.18.4	Advisory	The +, -, +=, and -= operators should not be applied to an expression of pointer type.	87 S - Use of pointer arithmetic.	87 S is waived when pointer arithmetic is needed to add an offset to a base address register for a hardware access.
			436 S - Declaration does not specify an array.	See R.18.1 reasoning for 436 S and 567 S.
			567 S - Pointer arithmetic is not on array.	

Table 4. Case-by-Case Deviations (continued)

MISRA-C Guideline [1]	Category[1]	Guideline Description[1]	LDRA Standard(s)[2]	Reasoning
R.21.2	Required	A reserved identifier or macro name shall not be declared.	219 S - User name starts with underscore.	219 S is only waived for compiler intrinsic calls such as: __disable_interrupts() __enable_interrupts() __byte_peripheral_32() __byte()

3 References

- Motor Industry Software Reliability Association (MISRA), *MISRA C:2012 Guidelines for the Use of the C Language in Critical Systems*, ISBN 978-1-906400-10-1, MISRA Limited, March 2013.
- Liverpool Data Research Associates, [MISRA-C:2012 Standards Model for C/C++](#), LDRA Tool Suite Installation Download
- Texas Instruments, [C2000 Compiler User's Guide](#)

IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated ("TI") technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, "TI Resources") are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI's provision of TI Resources does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI RESOURCES ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your non-compliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI's standard terms for semiconductor products (<http://www.ti.com/sc/docs/stdterms.htm>), [evaluation modules](#), and [samples](http://www.ti.com/sc/docs/sampterm.htm) (<http://www.ti.com/sc/docs/sampterm.htm>).

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2018, Texas Instruments Incorporated