Polyspace Code Verification

Developer Report for Project: polyspace

Report Author: ctreille

Polyspace Code Verification: Developer Report for Project: polyspace

by Report Author: ctreille

Published 19-Jan-2019 17:30:28

Verification Author(s): Polyspace
Polyspace Version(s): Polyspace Code Prover 10.0 (R2019a)
Project Version(s): 1.0

Table of Contents

Chapter 1. Polyspace Code Verification Summary

Table 1.1. Code Metrics Summary

Polyspace Code Metrics	Disabled
Pass/Fail	

Table 1.2. Coding Standard Summary - MISRA C:2004 Coding Standard

MISRA C:2004 Coding Standard	Enabled
Violations	37
Pass/Fail	

Table 1.3. Run-Time Checks Summary

Run-Time Checks	Enabled
Number of Red Checks	0
Number of Gray Checks	1
Number of Orange Checks	0
Number of Green Checks	13
Proven	100.0%
Pass/Fail	

Developer Name:

Date Reviewed:

Comments

Approved By:

Approved Date:

Chapter 2. Polyspace Run-Time Checks Statistics

Run-Time Checks Summary for polyspace - OR-414-1-developer

Globally Proven: 100.0%

File	Proven	Green	Red	Gray	Orange
OR-414-1-developer.c	100.0%	13	0	1	0
Total	**100.0%**	**13**	**0**	**1**	**0**

Percentage of code checked for run-time errors

Result Set	% Checked
polyspace - OR-414-1-developer	81%

Chapter 3. MISRA C:2004 Coding Standard

MISRA C:2004 Coding Standard Summary - Violations by File

File	Total
C:\qualkits_R2019a\iec\codeprover\tests\reporting\OR-414-1-developer.c	37
Total	**37**

MISRA C:2004 Coding Standard Violations

Table 3.1. C:\qualkits_R2019a\iec\codeprover\tests\reporting\OR-414-1-developer.c

ID	Rule	Message	Function	Line	Col	Jus	Severity	Status	Comment
1	20.1	Reserved identifiers, macros and functions in the standa rd library, shall not be defined, redefined or undefined. The macro 'offsetof' shall not be redefined.	File Scope	4	0	No	Unset	Unreviewed	
2	19.7	A function should be used in preference to a macro.	File Scope	4	8	No	Unset	Unreviewed	
13	6.3	Typedefs that indicate size and signedness should be used in place of the basic types.	File Scope	6	0	No	Unset	Unreviewed	
17	6.3	Typedefs that indicate size and signedness should be used in place of the basic types.	File Scope	7	0	No	Unset	Unreviewed	
31	18.1	All structure or union types shall be complete at the end of a translation unit.	File Scope	8	11	No	Unset	Unreviewed	
8	18.4	Unions shall not be used.	foo1()	10	4	No	Unset	Unreviewed	
28	18.1	All structure or union types shall be complete at the end of a translation unit.	File Scope	10	10	No	Unset	Unreviewed	
9	6.3	Typedefs that indicate size and signedness should be used in place of the basic types.	File Scope	15	0	No	Unset	Unreviewed	
33	8.1	Functions shall have prototype declarations and the proto type shall be visible at both the function definition and ca ll. Function 'foo' has no visible prototype at definition.	File Scope	15	4	No	Unset	Unreviewed	

ID	Rule	Message	Function	Line	Col	Jus	Severity	Status	Comment
37	8.10	All declarations and definitions of objects or functions at file scope shall have internal linkage unless external link age is required. Function 'foo' should have internal linkage	File Scope	15	4	No	Unset	Unreviewed	
23	6.3	Typedefs that indicate size and signedness should be used in place of the basic types.	File Scope	19	0	No	Unset	Unreviewed	
12	8.1	Functions shall have prototype declarations and the proto type shall be visible at both the function definition and ca ll. Function 'bar' has no visible prototype at definition.	File Scope	19	4	No	Unset	Unreviewed	
21	14.7	A function shall have a single point of exit at the end of t he function.	bar()	19	4	No	Unset	Unreviewed	
35	8.10	All declarations and definitions of objects or functions at file scope shall have internal linkage unless external link age is required. Function 'bar' should have internal linkage	File Scope	19	4	No	Unset	Unreviewed	
25	6.3	Typedefs that indicate size and signedness should be used in place of the basic types.	bar()	20	13	No	Unset	Unreviewed	
10	14.9	An if (expression) construct shall be followed by a compo und statement. The else keyword shall be followed by eith er a compound statement, or another if statement. An if (expression) construct shall be followed by a compo und statement.	bar()	21	4	No	Unset	Unreviewed	
29	13.2	Tests of a value against zero should be made explicit, unle ss the operand is effectively Boolean.	bar()	21	4	No	Unset	Unreviewed	
18	14.9	An if (expression) construct shall be followed by a compo und statement. The else keyword shall be followed by eith er a compound statement, or another if statement. The else keyword shall be followed by either a compound statement, or another if statement.	bar()	23	4	No	Unset	Unreviewed	
32	16.2	Functions shall not call themselves, either directly or indir ectly. Function bar shall not call itself, either directly or indirect ly.	bar()	24	16	No	Unset	Unreviewed	
22	8.1	Functions shall have prototype declarations and the proto type shall be visible at both the function definition and ca ll. Function 'func' has no visible prototype at definition.	File Scope	27	5	No	Unset	Unreviewed	
36	8.10	All declarations and definitions of objects or functions at file scope shall have internal linkage unless external link age is required.	File Scope	27	5	No	Unset	Unreviewed	

ID	Rule	Message	Function	Line	Col	Jus	Severity	Status	Comment
		Function 'func' should have internal linkage							
30	16.10	If a function returns error information, then that error information should be tested.	func()	28	7	No	Unset	Unreviewed	
34	16.10	If a function returns error information, then that error information should be tested.	func()	29	7	No	Unset	Unreviewed	
15	6.3	Typedefs that indicate size and signedness should be used in place of the basic types.	File Scope	33	0	No	Unset	Unreviewed	
26	6.3	Typedefs that indicate size and signedness should be used in place of the basic types.	File Scope	33	9	No	Unset	Unreviewed	
20	6.3	Typedefs that indicate size and signedness should be used in place of the basic types.	File Scope	35	0	No	Unset	Unreviewed	
11	6.3	Typedefs that indicate size and signedness should be used in place of the basic types.	File Scope	35	9	No	Unset	Unreviewed	
16	6.3	Typedefs that indicate size and signedness should be used in place of the basic types.	main()	38	4	No	Unset	Unreviewed	
27	6.3	Typedefs that indicate size and signedness should be used in place of the basic types.	main()	39	4	No	Unset	Unreviewed	
24	6.3	Typedefs that indicate size and signedness should be used in place of the basic types.	main()	40	4	No	Unset	Unreviewed	
3	4.2	Trigraphs shall not be used.	File Scope	46	60	No	Unset	Unreviewed	
5	4.2	Trigraphs shall not be used.	File Scope	47	35	No	Unset	Unreviewed	
4	4.2	Trigraphs shall not be used.	File Scope	47	47	No	Unset	Unreviewed	
14	13.7	Boolean operations whose results are invariant shall not be permitted. Expression is always false.	main()	49	12	No	Unset	Unreviewed	
6	4.2	Trigraphs shall not be used.	File Scope	49	19	No	Unset	Unreviewed	
7	4.2	Trigraphs shall not be used.	File Scope	53	8	No	Unset	Unreviewed	
19	16.10	If a function returns error information, then that error information should be tested.	main()	64	7	No	Unset	Unreviewed	

Chapter 4. Polyspace Run-Time Checks Results

Proven Run-Time Violations

No Proven Run-Time Violations checks were found.

Proven Unreachable Code Branches

Table 4.1. C:\qualkits_R2019a\iec\codeprover\tests\reporting\OR-414-1-developer.c

ID	Check	Function	Line	Col	Detail	Jus	Severity	Status	Comment
45	Unreachable code	main()	49	19	The section of code is unreachable or the condition is redundant. If-condition always evaluates to false at line 49 (column 12). Block ends at line 51 (column 4)	No	Unset	Unreviewed	

Unreachable Functions

No unreachable functions were found.

Unproven Run-Time Checks

No Unproven Run-Time Checks checks were found.

Chapter 5. Global Variables

Variable Checks for: polyspace - OR-414-1-developer

No variable checks were found in the code.

Chapter 6. Appendix 1 - Configuration Settings

Polyspace Settings

Option	Value
-author	Polyspace
-compiler	generic
-date	19/01/2019
-I	C:\qualkits_R2019a\iec\codeprover\tests\options-api\lib
-lang	C
-misra2	all-rules
-O2	-O2
-prog	polyspace
-results-dir	C:\qualkits_R2019a\iec\codeprover\tests\execution-folder-code-prover\reporting\OR-414-1-developer
-target	i386
-to	pass2
-verif-version	1.0

Analysis Assumptions

Assumption	Issuer
Nonfinite floats (infinities and NaNs) are not considered	Product
Results of floating-point arithmetic are rounded following the IEE754 rule: round to nearest, ties to even	Product
Structure fields are not volatile unless the entire structure is volatile-qualified	Product
Stack pointers can be safely dereferenced even outside the pointed variable's scope	Product
External pointers cannot be null. They point to allocated data of sufficient size for safe dereference	Product
Absolute addresses can be safely dereferenced	Product

Coding Standard Configuration

Table 6.1. MISRA C:2004 Coding Standard Configuration

Rule	Description	Mode	Comment	Enabled
1.1	All code shall conform to ISO 9899:1990 'Programming languages - C', amended and corrected by ISO/IEC 9899/COR1:1995, ISO/IEC 9899/AMD1:1995, and ISO/IEC 9899/COR2:1996.	required	-	yes
1.2	No reliance shall be placed on undefined or unspecified behaviour.	required	-	no
1.3	Multiple compilers and/or languages shall only be used if there is a common defined interface standard f or object code to which the language/compilers/assemblers conform.	required	-	no
1.4	The compiler/linker shall be checked to ensure that 31 character significance and case sensitivity are supp orted for external identifiers.	required	-	no
1.5	Floating point implementations should comply with a defined floating point standard.	advisory	-	no
2.1	Assembly language shall be encapsulated and isolated.	required	-	yes
2.2	source code shall only use /* ... */ style comments.	required	-	yes
2.3	The character sequence /* shall not be used within a comment.	required	-	yes
2.4	Sections of code should not be 'commented out'.	advisory	-	no
3.1	All usage of implementation-defined behaviour shall be documented.	required	-	no
3.2	The character set and the corresponding encoding shall be documented.	required	-	no
3.3	The implementation of integer division in the chosen compiler should be determined, documented and ta ken into account.	advisory	-	no
3.4	All uses of the #pragma directive shall be documented and explained.	required	-	yes
3.5	If it is being relied upon, the implementation-defined behaviour and packing of bitfields shall be documen ted.	required	-	no
3.6	All libraries used in production code shall be written to comply with the provisions of this document, and shall have been subject to appropriate validation.	required	-	no
4.1	Only those escape sequences which are defined in the ISO C standard shall be used.	required	-	yes
4.2	Trigraphs shall not be used.	required	-	yes
5.1	Identifiers (internal and external) shall not rely on the significance of more than 31 characters.	required	-	yes
5.2	Identifiers in an inner scope shall not use the same name as an identifier in an outer scope, and therefore hide that identifier.	required	-	yes
5.3	A typedef name shall be a unique identifier.	required	-	yes
5.4	A tag name shall be a unique identifier.	required	-	yes

Rule	Description	Mode	Comment	Enabled
5.5	No object or function identifier with static storage duration should be reused.	advisory	-	yes
5.6	No identifier in one name space should have the same spelling as an identifier in another name space, wi th the exception of structure and union member names.	advisory	-	yes
5.7	No identifier name should be reused.	advisory	-	yes
6.1	The plain char type shall be used only for the storage and use of character values.	required	-	yes
6.2	Signed and unsigned char type shall be used only for the storage and use of numeric values.	required	-	yes
6.3	Typedefs that indicate size and signedness should be used in place of the basic types.	advisory	-	yes
6.4	Bit fields shall only be defined to be of type unsigned int or signed int.	required	-	yes
6.5	Bit fields of type signed int shall be at least 2 bits long.	required	-	yes
7.1	Octal constants (other than zero) and octal escape sequences shall not be used.	required	-	yes
8.1	Functions shall have prototype declarations and the prototype shall be visible at both the function definiti on and call.	required	-	yes
8.2	Whenever an object or function is declared or defined, its type shall be explicitly stated.	required	-	yes
8.3	For each function parameter the type given in the declaration and definition shall be identical, and the ret urn types shall also be identical.	required	-	yes
8.4	If objects or functions are declared more than once their types shall be compatible.	required	-	yes
8.5	There shall be no definitions of objects or functions in a header file.	required	-	yes
8.6	Functions shall always be declared at file scope.	required	-	yes
8.7	Objects shall be defined at block scope if they are only accessed from within a single function.	required	-	yes
8.8	An external object or function shall be declared in one and only one file.	required	-	yes
8.9	An identifier with external linkage shall have exactly one external definition.	required	-	yes
8.10	All declarations and definitions of objects or functions at file scope shall have internal linkage unless exte rnal linkage is required.	required	-	yes
8.11	The static storage class specifier shall be used in definitions and declarations of objects and functions that have internal linkage.	required	-	yes
8.12	When an array is declared with external linkage, its size shall be stated explicitly or defined implicitly by initialization.	required	-	yes
9.1	All automatic variables shall have been assigned a value before being used.	required	-	yes
9.2	Braces shall be used to indicate and match the structure in the non-zero initialization of arrays and struct ures.	required	-	yes
9.3	In an enumerator list, the '=' construct shall not be used to explicitly initialise members other than the firs t, unless all items are explicitly initialised.	required	-	yes
10.1	The value of an expression of integer type shall not be implicitly converted to a different underlying type.	required	-	yes

Rule	Description	Mode	Comment	Enabled
10.2	The value of an expression of floating type shall not be implicitly converted to a different type.	required	-	yes
10.3	The value of a complex expression of integer type may only be cast to a type that is narrower and of the same signedness as the underlying type of the expression.	required	-	yes
10.4	The value of a complex expression of float type may only be cast to narrower floating type.	required	-	yes
10.5	If the bitwise operator ~ and << are applied to an operand of underlying type unsigned char or unsigned short, the result shall be immediatly cast to the underlying type of the operand.	required	-	yes
10.6	The 'U' suffix shall be applied to all constants of unsigned types.	required	-	yes
11.1	Conversion shall not be performed between a pointer to a function and any type other than an integral type.	required	-	yes
11.2	Conversion shall not be performed between a pointer to an object and any type other than an integral type, another pointer to object type or a pointer to void.	required	-	yes
11.3	A cast should not be performed between a pointer type and an integral type.	advisory	-	yes
11.4	A cast should not be performed between a pointer to object type and a different pointer to object type.	advisory	-	yes
11.5	A cast shall not be performed that removes any const or volatile qualification from the type addressed by a pointer.	required	-	yes
12.1	Limited dependence should be placed on C's operator precedence rules in expressions.	advisory	-	yes
12.2	The value of an expression shall be the same under any order of evaluation that the standard permits.	required	-	yes
12.3	The sizeof operator should not be used on expressions that contain side effects.	required	-	yes
12.4	The right hand operand of a logical && or || operator shall not contain side effects.	required	-	yes
12.5	The operands of a logical && or || shall be primary-expressions.	required	-	yes
12.6	The operands of a logical operators (&&, || and !) should be effectively Boolean. Expressions that are effectively Boolean should not be used as operands to operators other than (&&, || and !).	advisory	-	yes
12.7	Bitwise operators shall not be applied to operands whose underlying type is signed.	required	-	yes
12.8	The right hand operand of a shift operator shall lie between zero and one less than the width in bits of the underlying type of the left hand operand.	required	-	yes
12.9	The unary minus operator shall not be applied to an expression whose underlying type is unsigned.	required	-	yes
12.10	The comma operator shall not be used.	required	-	yes
12.11	Evaluation of constant unsigned integer expressions should not lead to wrap-around.	advisory	-	yes
12.12	The underlying bit representations of floating-point values shall not be used.	required	-	yes
12.13	The increment (++) and decrement (--) operators should not be mixed with other operators in an expression	advisory	-	yes
13.1	Assignment operators shall not be used in expressions that yield a Boolean value.	required	-	yes
13.2	Tests of a value against zero should be made explicit, unless the operand is effectively Boolean.	advisory	-	yes

Rule	Description	Mode	Comment	Enabled
13.3	Floating-point expressions shall not be tested for equality or inequality.	required	-	yes
13.4	The controlling expression of a for statement shall not contain any objects of floating type.	required	-	yes
13.5	The three expressions of a for statement shall be concerned only with loop control.	required	-	yes
13.6	Numeric variables being used within a for loop for iteration counting should not be modified in the body of the loop.	required	-	yes
13.7	Boolean operations whose results are invariant shall not be permitted.	required	-	yes
14.1	There shall be no unreachable code.	required	-	yes
14.2	All non-null statements shall either have at least one side effect however executed or cause control flow to change.	required	-	yes
14.3	Before preprocessing, a null statement shall only occur on a line by itself; it may be followed by a comme nt provided that the first character following the null statement is white-space character.	required	-	yes
14.4	The goto statement shall not be used.	required	-	yes
14.5	The continue statement shall not be used.	required	-	yes
14.6	For any iteration statement there shall be at most one break statement used for loop termination.	required	-	yes
14.7	A function shall have a single point of exit at the end of the function.	required	-	yes
14.8	The statement forming the body of a switch, while, do ... while or for statement shall be a compound state ment.	required	-	yes
14.9	An if (expression) construct shall be followed by a compound statement. The else keyword shall be follow ed by either a compound statement, or another if statement.	required	-	yes
14.10	All if ... else if constructs should contain a final else clause.	required	-	yes
15.0	A switch statement shall conform to MISRA-C syntax.	required	-	yes
15.1	A switch label shall only be used when the most closely-enclosing compound statement is the body of a sw itch statement.	required	-	yes
15.2	An unconditional break statement shall terminate every non-empty switch clause.	required	-	yes
15.3	The final clause of a switch statement shall be the default clause.	required	-	yes
15.4	A switch expression should not represent a value that is effectively Boolean.	required	-	yes
15.5	Every switch statement shall have at least one case clause.	required	-	yes
16.1	Functions shall not be defined with variable numbers of arguments.	required	-	yes
16.2	Functions shall not call themselves, either directly or indirectly.	required	-	yes
16.3	Identifiers shall be given for all of the parameters in a function prototype declaration.	required	-	yes
16.4	The identifiers used in the declaration and definition of a function shall be identical.	required	-	yes
16.5	Functions with no parameters shall be declared with parameter type void.	required	-	yes
16.6	The number of arguments passed to a function shall match the number of parameters.	required	-	yes

Rule	Description	Mode	Comment	Enabled
16.7	A pointer parameter in a function prototype should be declared as pointer to const if the pointer is not us ed to modify the addressed object.	advisory	-	yes
16.8	All exit paths from a function with non-void return type shall have an explicit return statement with an ex pression.	required	-	yes
16.9	A function identifier shall only be used with either a preceding &, or with a parenthesised parameter list, which may be empty.	required	-	yes
16.10	If a function returns error information, then that error information should be tested.	required	-	yes
17.1	Pointer arithmetic shall only be applied to pointers that address an array or array element.	required	-	yes
17.2	Pointer subtraction shall only be applied to pointers that address elements of the same array.	required	-	yes
17.3	>, >=, <, <= shall not be applied to pointer types except where they point to the same array.	required	-	yes
17.4	Array indexing shall be the only allowed form of pointer arithmetic.	required	-	yes
17.5	The declaration of objects should contain no more than 2 levels of pointer indirection.	advisory	-	yes
17.6	The address of an object with automatic storage shall not be assigned to an object that may persist after t he object has ceased to exist.	required	-	yes
18.1	All structure or union types shall be complete at the end of a translation unit.	required	-	yes
18.2	An object shall not be assigned to an overlapping object.	required	-	yes
18.3	An area of memory shall not be reused for unrelated purposes.	required	-	no
18.4	Unions shall not be used.	required	-	yes
19.1	#include statements in a file shall only be preceded by other pre-processor directives or comments.	advisory	-	yes
19.2	Non-standard characters should not occur in header file names in #include directives.	advisory	-	yes
19.3	The #include directive shall be followed by either a <filename> or "filename" sequence.	required	-	yes
19.4	C macros shall only expand to a braced initialiser, a constant, a parenthesised expression,a type qualifier, a storage class specifier, or a do-while-zero construct.	required	-	yes
19.5	Macros shall not be #define'd and #undef'd within a block.	required	-	yes
19.6	#undef shall not be used.	required	-	yes
19.7	A function should be used in preference to a macro.	advisory	-	yes
19.8	A function-like macro shall not be invoked without all of its arguments.	required	-	yes
19.9	Arguments to a function-like macro shall not contain tokens that look like pre-processing directives.	required	-	yes
19.10	In the definition of a function-like macro each instance of a parameter shall be enclosed in parentheses u nless it is used as the operand of # or ##.	required	-	yes
19.11	All macro identifiers in preprocessor directives shall be defined before use, except in #ifdef and #ifndef pr eprocessor directives and the defined() operator.	required	-	yes
19.12	There shall be at most one occurrence of the # or ## pre-processor operators in a single macro definition.	required	-	yes

Rule	Description	Mode	Comment	Enabled
19.13	The # and ## preprocessor operators should not be used.	advisory	-	yes
19.14	The defined pre-processor operator shall only be used in one of the two standard forms.	required	-	yes
19.15	Precautions shall be taken in order to prevent the contents of a header file being included twice.	required	-	yes
19.16	Preprocessing directives shall be syntactically meaningful even when excluded by the preprocessor.	required	-	yes
19.17	All #else, #elif and #endif preprocessor directives shall reside in the same file as the #if or #ifdef directive to which they are related.	required	-	yes
20.1	Reserved identifiers, macros and functions in the standard library, shall not be defined, redefined or unde fined.	required	-	yes
20.2	The names of standard library macros, objects and functions shall not be reused.	required	-	yes
20.3	The validity of values passed to library functions shall be checked.	required	-	yes
20.4	Dynamic heap memory allocation shall not be used.	required	-	yes
20.5	The error indicator errno shall not be used.	required	-	yes
20.6	The macro offsetof, in library <stddef.h>, shall not be used.	required	-	yes
20.7	The setjmp macro and the longjmp function shall not be used.	required	-	yes
20.8	The signal handling facilities of <signal.h> shall not be used.	required	-	yes
20.9	The input/output library <stdio.h> shall not be used in production code.	required	-	yes
20.10	The library functions atof, atoi and atol from library <stdlib.h> shall not be used.	required	-	yes
20.11	The library functions abort, exit, getenv and system from library <stdlib.h> shall not be used.	required	-	yes
20.12	The time handling functions of library <time.h> shall not be used.	required	-	yes
21.1	Minimisation of run-time failures shall be ensured by the use of at least one tool/technique.	required	-	yes

Chapter 7. Appendix 2 - Definitions

Table 7.1. Abbreviations

Abbreviation	Definition
Col	Column
Jus	Justified
SQO	Software Quality Objectives
NA	Not Available