



Sicherungskomponente für
Autonome Mobile Systeme

Eine Kooperation zwischen
DFKI-Labor Bremen • Leuze lumiflex • Universität Bremen

Maßnahmen zur Prüfung der Einhaltung der MISRA-Richtlinien

Zusammenfassung

Die MISRA-Richtlinien sind wesentlicher Bestandteil der Programmierrichtlinien im Rahmen des SAMS-Projektes. Dieses Dokument beschreibt die Planung und Durchführung der Prüfung der Einhaltung dieser Richtlinien.

<i>Projektbezeichnung</i>	SAMS
<i>Verantwortlich</i>	Christoph Lüth, Hennes Märtins
<i>Erstellt am</i>	05.01.2009
<i>Version</i>	1.1
<i>Bearbeitungszustand</i>	fg.
<i>Revision</i>	4326
<i>Letzte Änderung</i>	08.10.2009
<i>Dokumentablage</i>	Projektdokumente/Validierung/Verifikation/misra-check.tex

Änderungsliste

05.01.09 – Version: 1.0 – Bearbeiter: H. Märtns

Initiale Version.

19.01.09 – Version: 1.1 – Bearbeiter: C. Lüth

Überarbeitet, anderes Bewertungssystem mit Maßnahme, Priorität und Komplexität eingeführt.

Prüfverzeichnis

25.09.2009 – Version: 1.1 – Prüfer: TÜV (P. Supavatanakul)

Neuer Produktzustand: **fg**.

Siehe Prüfbericht (Technical Report no. LF82764T)

Inhaltsverzeichnis

1	Überblick	4
1.1	Maßnahmen	4
1.2	Komplexität und Priorität der Maßnahme	4
2	MISRA-Regeln	5

1 Überblick

Die MISRA-Richtlinien MISRA (2004) bestehen aus mehr als hundert einzelnen *Regeln*. Ihre Einhaltung wird im Rahmen des Projektes SAMS durch *Maßnahmen* gewährleistet, die in diesem Dokument erläutert werden. Die Maßnahmen lassen sich grob unterteilen in Prüfungen, die durch ein im Rahmen des Projektes entwickeltes Werkzeug geleistet werden, sowie andere Maßnahmen, namentlich Code Review und Testen.

Die Tabelle in Abschnitt 2 enthält für jede Regel die Prüfmaßnahme, sowie eine Abschätzung der Priorität und Komplexität der Prüfung; diese Dokument dient damit auch der Planung der Implementation des Prüfwerkzeugs, und wird fortschreitend mit dessen Implementationsstand aktualisiert.

1.1 Maßnahmen

Die Maßnahmen sind in drei Kategorien klassifiziert; Maßnahmen der Kategorie A werden durch das Werkzeug implementiert, Maßnahmen der Kategorie B sollen durch das Werkzeug implementiert werden, und Maßnahme der Kategorie C werden extern implementiert. Die Tests für Maßnahmen der Kategorie A werden soweit sinnvoll mit der MISRA-C *exemplar suite* durchgeführt.

Kategorie			Maßnahme zur Einhaltung der Regel
A	B	C	
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		kann nicht auftreten Werkzeug muss prüfen (geplant)
<input checked="" type="checkbox"/>			kann nicht auftreten, Werkzeug prüft (implementiert und getestet)
		<input checked="" type="checkbox"/>	kann nicht auftreten (extern sichergestellt)
	<input type="checkbox"/>		Werkzeug muss prüfen (geplant)
	<input checked="" type="checkbox"/>		Werkzeug prüft (implementiert)
<input checked="" type="checkbox"/>			Werkzeug prüft (implementiert und getestet)
		<input checked="" type="checkbox"/>	Wird durch CodeReview sichergestellt
		<input checked="" type="checkbox"/>	Wird durch Testen sichergestellt

1.2 Komplexität und Priorität der Maßnahme

Komplexität und Priortität der Maßnahme werden nur für die Planung der Implementierung der Prüfmaßnahmen oder deren Durchführung benötigt. Regeln der Priorität A sind dabei relevant für semantische Integrität (d.h. ihre Einhaltung wird in der Verifikation vorausgesetzt), oder sie sind tückische, schwer zu findende Fehlerquelle (wie arithmetische Konversionen); Regeln der Priorität C sind seltene Fehlerquellen (wie oktale Konstanten).

Komplexität		Priorität	
1	trivial	A	Höchste Priorität
2	einfach	B	Mittlere Priorität
3	anspruchsvoll	C	Niedrige Priorität

2 MISRA-Regeln

In der folgenden Tabelle bezeichnen die letzten drei Spalten die Maßnahme, ihr Komplexität, und ihre Priorität. Die Anmerkungen finden sich hinter den Regeln auf Seite 13.

Environment

1.1 (req)	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 (<i>siehe Anmerkung 1</i>).	⊖		
1.2 (req)	No reliance shall be placed on undefined or unspecified behaviour (<i>siehe Anmerkung 2</i>).	⊗	2	A
1.3 (req)	Multiple compilers and/or languages shall only be used if there is a common defined interface standard for object code to which the languages/compilers/assemblers conform.	⊖	1	C
1.4 (req)	The compiler/linker shall be checked to ensure that 31 character significance and case sensitivity are supported for external identifiers.	Ⓜ	1	C
1.5 (adv)	Floating-point implementations should comply with a defined floating-point standard (<i>siehe Anmerkung 3</i>).	⊖	1	B



Language extensions

2.1 (req)	Assembly language shall be encapsulated and isolated.	⊗	1	B
2.2 (req)	Source code shall only use /* ... */ style comments.	✓	2	B
2.3 (req)	The character sequence /* shall not be used within a comment.	✓	2	B
2.4 (adv)	Sections of code should not be “commented out”.	Ⓜ	2	B








Documentation

3.1 (req)	All usage of implementation-defined behaviour shall be documented.	Ⓜ	3	B
3.2 (req)	The character set and the corresponding encoding shall be documented (<i>siehe Anmerkung 4</i>).	Ⓜ	1	C
3.3 (adv)	The implementation of integer division in the chosen compiler should be determined, documented and taken into account.	Ⓜ	1	B
3.4 (req)	All uses of the #pragma directive shall be documented and explained.	⊗	1	C
3.5 (req)	The implementation-defined behaviour and packing of bitfields shall be documented if being relied upon.	⊗	1	C
3.6 (req)	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 (<i>siehe Anmerkung 5</i>).	⊗	1	B






Character sets

4.1 (req)	Only those escape sequences that are defined in the ISO C standard shall be used.		1	C
4.2 (req)	Trigraphs shall not be used.		1	C


Identifiers

5.1 (req)	Identifiers (internal and external) shall not rely on the significance of more than 31 characters.		1	B
5.2 (req)	Identifiers in an inner scope shall not use the same name as an identifier in an outer scope, and therefore hide that identifier.		1	B
5.3 (req)	A typedef name shall be a unique identifier.		2	B
5.4 (req)	A tag name shall be a unique identifier.		2	B
5.5 (adv)	No object or function identifier with static storage duration should be reused.		2	B
5.6 (adv)	No identifier in one name space should have the same spelling as an identifier in another name space, with the exception of structure member and union member names.		2	B
5.7 (adv)	No identifier name should be reused.		2	B













Types

6.1 (req)	The plain char type shall be used only for storage and use of character values.		2	B
6.2 (req)	signed and unsigned char type shall be used only for the storage and use of numeric values.		2	B
6.3 (adv)	typedefs that indicate size and signedness should be used in place of the basic numerical types.		2	B
6.4 (req)	Bit fields shall only be defined to be of type unsigned int or signed int.		1	C
6.5 (req)	Bit fields of type signed type shall be at least 2 bits long.		1	C




Constants

7.1 (req)	Octal constants (other than zero) and octal escape sequences shall not be used.		2	C
-----------	---	---	---	---

Declarations and definitions

8.1 (req)	Functions shall have prototype declarations and the prototype shall be visible at both the function definition and call.		1	A
8.2 (req)	Whenever an object or function is declared or defined, its type shall be explicitly stated.		1	A
8.3 (req)	For each function parameter the type given in the declaration and definition shall be identical, and the return types shall also be identical.		2	B
8.4 (req)	If objects or functions are declared more than once their types shall be compatible.		2	B
8.5 (req)	There shall be no definitions of objects or functions in a header file.		2	B
8.6 (req)	Functions shall be declared at file scope.		1	B
8.7 (req)	Objects shall be defined at block scope if they are only accessed from within a single function.		2	B
8.8 (req)	An external object or function shall be declared in one and only one file.		2	B
8.9 (req)	An identifier with external linkage shall have exactly one external definition.		2	B
8.10 (req)	All declarations and definitions of objects or functions at file scope shall have internal linkage unless external linkage is required.		3	B
8.11 (req)	The static storage class specifier shall be used in definitions and declarations of objects and functions that have internal linkage.		3	B
8.12 (req)	When an array is declared with external linkage, its size shall be stated explicitly or defined implicitly by initialisation.		2	B

Initialisation

9.1 (req)	All automatic variables shall have been assigned a value before being used.		3	A
9.2 (req)	Braces shall be used to indicate and match the structure in the non-zero initialisation of arrays and structures.		1	B
9.3 (req)	In an enumerator list, the “=” construct shall not be used to explicitly initialise members other than the first, unless all items are explicitly initialised.		1	B

Arithmetic type conversions

10.1 (req)	The value of an expression of integer type shall not be implicitly converted to a different underlying type if: <ul style="list-style-type: none"> a) it is not a conversion to a wider integer type of the same signedness, or b) the expression is complex, or c) the expression is not constant and is a function argument, or d) the expression is not constant and is a return expression. 	✓	2	A
10.2 (req)	The value of an expression of floating type shall not be implicitly converted to a different type if: <ul style="list-style-type: none"> a) it is not a conversion to a wider floating type, or b) the expression is complex, or c) the expression is a function argument, or d) the expression is a return expression. 	✓	1	A
10.3 (req)	The value of a complex expression of integer type shall only be cast to a type of the same signedness that is no wider than the underlying type of the expression.	✓	2	A
10.4 (req)	The value of a complex expression of floating type shall only be cast to a floating type which is narrower or of the same size.	✓	1	A
10.5 (req)	If the bitwise operators \sim and \ll are applied to an operand of underlying type unsigned char or unsigned short, the result shall be immediately cast to the underlying type of the operand.	✓	2	A
10.6 (req)	A “U” suffix shall be applied to all constants of unsigned type.	✓	2	B

Pointer type conversions

11.1 (req)	Conversions shall not be performed between a pointer to a function and any type other than an integral type.	✓	1	C
11.2 (req)	Conversions shall not be performed between a pointer to object and any type other than an integral type, another pointer to object type or a pointer to void.	✓	1	C
11.3 (adv)	A cast should not be performed between a pointer type and an integral type.	✓	1	C
11.4 (adv)	A cast should not be performed between a pointer to object type and a different pointer to object type.	✓	1	C

11.5 (req)	A cast shall not be performed that removes any const or volatile qualification from the type addressed by a pointer.		1	C
------------	--	--	---	---

Expressions

12.1 (adv)	Limited dependence should be placed on C's operator precedence rules in expressions.		3	-
12.2 (req)	The value of an expression shall be the same under any order of evaluation that the standard permits.		3	A
12.3 (req)	The sizeof operator shall not be used on expressions that contain side effects.		3	B
12.4 (req)	The right-hand operand of a logical && or operator shall not contain side effects.		3	B
12.5 (req)	The operands of a logical && or shall be primary-expressions.		3	B
12.6 (adv)	The operands of logical operators (&&, and !) should be effectively Boolean. Expressions that are effectively Boolean should not be used as operands to operators other than (&&, , !, =, ==, !=, and ?:).		3	B
12.7 (req)	Bitwise operators shall not be applied to operands whose underlying type is signed.		2	B
12.8 (req)	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.		3	B
12.9 (req)	The unary minus operator shall not be applied to an expression whose underlying type is unsigned.		2	B
12.10 (req)	The comma operator shall not be used.		1	A
12.11 (adv)	Evaluation of constant unsigned integer expressions should not lead to wrap-around. (<i>siehe Anmerkung 8</i>)		3	B
12.12 (req)	The underlying bit representations of floating-point values shall not be used.		3	C
12.13 (adv)	The increment (++) and decrement (--) operators should not be mixed with other operators in an expression.		1	B

Control statement expressions

13.1 (req)	Assignment operators shall not be used in expressions that yield a Boolean value.		1	A
13.2 (adv)	Tests of a value against zero should be made explicit, unless the operand is effectively Boolean.		3	A
13.3 (req)	Floating-point expressions shall not be tested for equality or inequality.		1	A
13.4 (req)	The controlling expression of a for statement shall not contain any objects of floating type.		1	A

13.5 (req)	The three expressions of a for statement shall be concerned only with loop control.	✓	1	A
13.6 (req)	Numeric variables being used within a for loop for iteration counting shall not be modified in the body of the loop.	✓	3	A
13.7 (req)	Boolean operations whose results are invariant shall not be permitted.	✓	3	B

Control flow

14.1 (req)	There shall be no unreachable code.	✘	3	-
14.2 (req)	All non-null statements shall either: a) have at least one side effect however executed, or b) cause control flow to change.	✓	3	B
14.3 (req)	Before preprocessing, a null statement shall only occur on a line by itself; it may be followed by a comment provided that the first character following the null statement is a white-space character.	✓	3	C
14.4 (req)	The goto statement shall not be used.	✓	1	A
14.5 (req)	The continue statement shall not be used.	✓	1	A
14.6 (req)	For any iteration statement there shall be at most one break statement used for loop termination.	✓	2	B
14.7 (req)	A function shall have a single point of exit at the end of the function.	✓	1	A
14.8 (req)	The statement forming the body of a switch, while, do ... while or for statement be a compound statement.	✓	1	C
14.9 (req)	An if (expression) construct shall be followed by a compound statement. The else keyword shall be followed by either a compound statement, or another if statement.	✓	1	C
14.10 (req)	All if ... else if constructs shall be terminated with an else clause.	✓	1	B

Switch statements

15.1 (req)	A switch label shall only be used when the most closely-enclosing compound statement is the body of a switch statement.	✘	1	A
15.2 (req)	An unconditional break statement shall terminate every non-empty switch clause.	✘	1	A
15.3 (req)	The final clause of a switch statement shall be the default clause.	✘	1	A
15.4 (req)	A switch expression shall not represent a value that is effectively Boolean.	✘	1	A
15.5 (req)	Every switch statement shall have at least one case clause.	✘	1	A

Functions

16.1 (req)	Functions shall not be defined with variable numbers of arguments.	✓	1	A
16.2 (req)	Functions shall not call themselves, either directly or indirectly.	✓	3	A
16.3 (req)	Identifiers shall be given for all of the parameters in a function prototype declaration.	✗	1	B
16.4 (req)	The identifiers used in the declaration and definition of a function shall be identical.	✓	1	B
16.5 (req)	Functions with no parameters shall be declared and defined with the parameter list void.	✓	1	B
16.6 (req)	The number of arguments passed to a function shall match the number of parameters.	✗	2	A
16.7 (adv)	A pointer parameter in a function prototype should be declared as pointer to const if the pointer is not used to modify the addressed object.	✓	3	B
16.8 (req)	All exit paths from a function with non-void return type shall have an explicit return statement with an expression.	✓	3	A
16.9 (req)	A function identifier shall only be used with either a preceding &, or with a parenthesised parameter list, which may be empty.	✓	1	B
16.10 (req)	If a function returns error information, then that error information shall be tested.	Ⓜ	3	-

Pointers and arrays

17.1 (req)	Pointer arithmetic shall only be applied to pointers that address an array or array element.	✓	3	C
17.2 (req)	Pointer subtraction shall only be applied to pointers that address elements of the same array.	✓	3	C
17.3 (req)	>, >=, <, <= shall not be applied to pointer types except where they point to the same array.	✗	3	C
17.4 (req)	Array indexing shall be the only allowed form of pointer arithmetic.	✓	3	A
17.5 (adv)	The declaration of objects should contain no more than 2 levels of pointer indirection.	✓	1	B
17.6 (req)	The address of an object with automatic storage shall not be assigned to another object that may persist after the first object has ceased to exist.	○	3	B

Structures and unions

18.1 (req)	All structure or union types shall be complete at the end of a translation unit. (<i>siehe Anmerkung 9</i>)	–	3	B
18.2 (req)	An object shall not be assigned to an overlapping object. (<i>siehe Anmerkung 7</i>)	✗	3	B

18.3 (req)	An area of memory shall not be reused for unrelated purposes.			C
18.4 (req)	Unions shall not be used.	✓	1	B

Preprocessing directives

19.1 (adv)	#include statements in a file should only be preceded by other preprocessor directives or comments.	✓	3	B
19.2 (adv)	Non-standard characters should not occur in header file names in #include directives.	✓	3	B
19.3 (req)	The #include directive shall be followed by either a <filename> or "filename" sequence. (siehe Anmerkung 10)	✓	3	B
19.4 (req)	C macros shall only expand to a braced initialiser, a constant, a string literal, a parenthesised expression, a type qualifier, a storage class specifier, or a do-while-zero construct. (siehe Anmerkung 11)	✓	3	B
19.5 (req)	Macros shall not be #define'd or #undef'd within a block.	✓		B
19.6 (req)	#undef shall not be used.	✓	2	B
19.7 (adv)	A function should be used in preference to a function-like macro.	✓	3	B
19.8 (req)	A function-like macro shall not be invoked without all of its arguments.	✓	3	C
19.9 (req)	Arguments to a function-like macro shall not contain tokens that look like preprocessing directives.	○	3	C
19.10 (req)	In the definition of a function-like macro each instance of a parameter shall be enclosed in parentheses unless it is used as the operand of # or ##.	✓	3	B
19.11 (req)	All macro identifiers in preprocessor directives shall be defined before use, except in #ifdef and #ifndef preprocessor directives and the defined() operator.	✓	3	B
19.12 (req)	There shall be at most one occurrence of the # or ## preprocessor operators in a single macro definition.	✓	3	B
19.13 (adv)	The # and ## preprocessor operators should not be used.	✓	3	B
19.14 (req)	The defined preprocessor operator shall only be used in one of the two standard forms.	✓	3	B
19.15 (req)	Precautions shall be taken in order to prevent the contents of a header file being included twice. (siehe Anmerkung 12)	✓	3	B
19.16 (req)	Preprocessing directives shall be syntactically meaningful even when excluded by the preprocessor.	✓	3	C
19.17 (req)	All #else, #elif and #endif preprocessor directives shall reside in the same file as the #if or #ifdef directive to which they are related.	✓	3	B

Standard libraries

20.1 (req)	Reserved identifiers, macros and functions in the standard library, shall not be defined, redefined or undefined.	⊙	3	C
20.2 (req)	The names of standard library macros, objects and functions shall not be reused.	✓	3	C
20.3 (req)	The validity of values passed to library functions shall be checked.	Ⓜ	3	B
20.4 (req)	Dynamic heap memory allocation shall not be used.	✓	2	A
20.5 (req)	The error indicator errno shall not be used.	✓	2	B
20.6 (req)	The macro offsetof, in library <stddef.h>, shall not be used.	✓	2	A
20.7 (req)	The setjmp macro and the longjmp function shall not be used.	✓	2	A
20.8 (req)	The signal handling facilities of <signal.h> shall not be used.	✗	2	A
20.9 (req)	The input/output library <stdio.h> shall not be used in production code.	✓	2	A
20.10 (req)	The library functions atof, atoi and atol from library <stdlib.h> shall not be used.	✓	2	A
20.11 (req)	The library functions abort, exit, getenv and system from library <stdlib.h> shall not be used.	✓	2	A
20.12 (req)	The time handling functions of library <time.h> shall not be used.	✗	2	A

Run-time failures

21.1 (req)	<p>Minimisation of run-time failures shall be ensured by the use of at least one of:</p> <ul style="list-style-type: none"> a) static analysis tools/techniques; b) dynamic analysis tools/techniques; c) explicit coding of checks to handle run-time faults. <p>(siehe Anmerkung 6)</p>	⊖	3	A
------------	--	---	---	---

Anmerkungen

1. Im Projekt wird ISO 9899:1999 verwendet; Einhaltung wird durch andere Werkzeuge sichergestellt.
2. undefiniertes und unspezifiziertes Verhalten wird durch die Beschränkung auf eine Sprachteilmenge verhindert. Deren Einhaltung muss von Werkzeug geprüft werden.
3. Verwendete Bücherei (gcc mathematical library) entspricht IEEE 754.
4. Wird durch Benutzung von Doxygen sichergestellt.
5. Es wird nur die Standard-Bücherei verwendet.

6. Wird durch Verifikation und Test behandelt.
7. “It is impossible to create overlapping objects in C without using non-standard features” (MISRA C exemplar suite).
8. Wird nicht geprüft für nur im Präprozessor evaluierte Ausdrücke (Argumente von `#if` u.ä.).
9. Einhaltung wird durch den C-Compiler (GCC) sichergestellt.
10. Es wird konservativ geprüft, das heißt wenn der Headername nicht explizit erkannt wird, dann wird gewarnt. Dann muss aber kein Fehler vorliegen, es kann nämlich sein, dass die Tokensequence sich zu einem Headername auflöst.
11. Es wird konservativ geprüft, ein `do-while`-Konstrukt oder ein geklammerter Ausdruck wird nicht komplett auflöst, es wird aber gewarnt, dass hier eine Fehlerquelle vorliegen könnte.
12. Es wird konservativ geprüft, ob die erste Zeile einer eingebundenen Header-Datei ein `#ifndef` und die zweite ein `#define` ist.

Literatur

[MISRA 2004] *MISRA-C: 2004. Guidelines for the use of the C language in critical systems.*
2004