CODE TIME TECHNOLOGIES

Abassi RTOS

MISRA-C:2004

Compliance Report

Disclaimer

Table of Contents

1 INTRODUCTION 5

2 OVERVIEW 6

3 BUILD OPTIONS 7

3.1 OS_LOGGING_TYPE 7
3.2 OS_RUNTIME 7
3.3 OS_SEARCH_ALGO 7

4 RULES WITH ISSUES 8

4.1 REQUIRED RULES 8
4.2 OPTIONAL RULES 8

5 RULE SET 9

List of Tables

TABLE 4-1 REQUIRED RULES LIST 8
TABLE 4-2 OPTIONAL RULES LIST 8

1 Introduction

This document is a report on the compliance of the Abassi RTOS to the MISRA-C:2004 coding standard. The first section describes the techniques used to verify the MISRA-C:2004 compliance of Abassi. The second section gives a list of build options that affect the MISRA-C:2004 compliance of Abassi, and their impact. The third section gives an overview of all rules that could be broken due to the setting of build options. And, finally, the last section enumerates each rule and sometime gives a description of the limitations of the rule.

2 Overview

The MISRA-C:2004 compliance was verified using IAR's Embedded Workbench for the ARM processor family, and Texas Instrument's Code Composer Studio for the MSP430.

The Abassi RTOS is not built using individual function and/or files. A single "C" file holds all the code for the RTOS, and build options control which features are included in the build through the use of "C" pre-processor `#if` statements. As there are a large number of build options combinations, the code submitted to the MISRA compliance checking was built using each of the Code Time Technologies 40-plus test configurations, which have been verified to perform full code coverage. Therefore, using all the test configurations, the whole code of Abassi was checked for compliance.

3 Build Options

There are some settings through the build options that void the MISRA compliance of Abassi. These build options are listed in this section.

3.1 OS_LOGGING_TYPE

The build option `OS_LOGGING_TYPE` must be set to a value of zero otherwise Abassi is not MISRA-C:2004 compliant. As stated in the Abassi User Guide, the logging facilities are targeted for debugging an application during the development, as it is quite intrusive. Logging should not be part of production software.

3.2 OS_RUNTIME

The build option `OS_RUNTIME` must be set to a negative value otherwise Abassi is not MISRA-C:2004 compliant. Setting the build option `OS_RUNTIME` to a non-negative value will enable the compile time creation of descriptors. The static creation of descriptors uses the "C" pre-processor in ways that do not comply with MISRA-C:2004.

3.3 OS_SEARCH_ALGO

The build option `OS_SEARCH_ALGO` must be set to a non-positive value otherwise Abassi is not MISRA-C:2004 compliant. Setting the build option `OS_SEARCH_ALGO` to a positive value was deemed to reduce the reliability of Abassi as two look-up tables are used to determine the next task to run. Holding redundant information in a second table creates a search algorithm with two points of failures instead of a single one. No effort was spent to make the double table search algorithm to comply with MISRA-C:2004.

4 Rules with issues

This section lists the rules that could make Abassi non-compliant. Refer to the next sections for a full description of the possible issues.

4.1 Required rules

Table 4-1 Required rules list

Rule	Description
8.7	Port dependent, can be fulfilled at a cost on the code footprint
10.1	Non-compliant when the build option `OS_SEARCH_ALGO` is positive
10.6	Non-compliant when the build option `OS_SEARCH_ALGO` is positive
12.4	The components `TSKstate()`, `TSKisRdy()`, `TSKisBlk()` and `TSKisSusp()` do not adhere to this rule. Not using any of these components fulfills the compliance.
12.7	Non-compliant when the build option `OS_SEARCH_ALGO` is positive
12.10	Non-compliant when the build option `OS_EVENTS` is non-zero
14.7	Non-compliant when the build option `OS_LOGGING_TYPE` is non-zero
16.1	Non-compliant when the build option `OS_LOGGING_TYPE` is non-zero
17.4	Non-compliant when the build option `OS_MAILBOX` is non-zero
19.4	Non-compliant when the build option `OS_RUNTIME` is non-negative. Also, when `OS_RUNTIME` is negative, some ports do not comply, but compliance can be fulfilled through minor small code changes.
19.12	Non-compliant when the build option `OS_RUNTIME` is non-negative
19.13	Non-compliant when the build option `OS_RUNTIME` is non-negative
20.4	Only compliant when the build options `OS_STATIC_XXX` are non-zero
20.9	Non-compliant when the build option `OS_LOGGING_TYPE` is non-zero
21.1	Clause c) should not be relied on

4.2 Optional rules

Table 4-2 Optional rules list

Rule	Description
6.3	The native “C” data types were chosen by design
11.3	Abassi cannot comply with this rule due to the use of the `intptr_t` data type
12.13	Not adhering to this rule delivers a more real-time efficient code
19.1	Port dependent
19.7	Many components are macros

5 Rule Set

Rule 1.1 (Required) 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.

Compliance: YES

Notes: The data type `intptr_t` defined in the library `stdint.h` is used as an argument to the kernel and also as the data type for the mailboxes. The data type `intptr_t` is standardized in C99, but not previous versions. This does not remove the compliance of the Kernel code with all previous "C" standards as an equivalent data type to `intptr_t` could have been `typedef` in `Abassi.h` for the same purpose.

Rule 1.2 (Required) No reliance shall be placed on undefined or unspecified behavior.

Compliance: YES

Notes:

Rule 1.3 (Required) Multiple compilers and/or languages shall only be used if there is a common defined interface standard for object code to which the language / compilers / assemblers conform.

Compliance: YES

Notes: By its nature and target market, the Abassi Kernel is multi-platform, multi-compiler and the API remains the same no matter what is the target the processor or compiler.

Rule 1.4 (Required) The compiler / linker shall be checked to ensure that 31 character significance and case sensitivity are supported for external identifiers.

Compliance: YES

Notes: All identifiers in Abassi are shorter than 31 characters. Most of Abassi's identifiers are unique over the first 7 characters.

Rule 1.5 (Advisory) Floating-point implementations should comply with a defined floating-point standard.

Compliance: YES

Notes: Floating point numbers are not used in Abassi.

Rule 2.1 (Advisory) Assembler language shall be encapsulated and isolated.

Compliance: YES

Notes: All assembly code is located in the file `Abassi_PROCESSOR_COMPILER.ext`, which is specific to the target processor and the target compiler / assembler / linker suite.

Rule 2.2 (Required) Source code shall only use `/* ... */` style comments.

Compliance: YES

Notes:

Rule 2.3 (Required) The character sequence `/*` shall not be used within a comment.

Compliance: YES

Notes:

Rule 2.4 (Advisory) Sections of code should not be commented out.

Compliance: YES

Notes:

Rule 3.1 (Required) All usage of implementation-defined behavior shall be documented.

Compliance: YES

Notes: The code is fully documented, where there is almost one comment per line of code.

Rule 3.2 (Required) The character set and the corresponding encoding shall be documented.

Compliance: YES

Notes: No special character set is used by the Abassi code.

Rule 3.3 (Advisory) The implementation of integer division in the chosen compiler should be determined, documented, and taken into account.

Compliance: YES

Notes: Integer division in Abassi only involves positive numbers divided by 1, 2, 4 or 8 and the division operation can be a real arithmetic division or a simple shift right, with no remainder.

Rule 3.4 (Required) All uses of the `#pragma` directive shall be documented and explained.

Compliance: YES

Notes: `#pragma` are only use in two ports and they are commented where used.

Rule 3.5 (Required) If it is being relied upon, the implementation-defined behavior and packing of bit fields shall be documented.

Compliance: YES

Notes: Bit fields are not used in Abassi.

Rule 3.6 (Required) 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.

Compliance: N/A

Notes:

Rule 4.1 (Required) Only those escape sequences that are defined in the ISO C standard shall be used.

Compliance: YES

Notes: Only the logging facilities (when the build option `OS_LOGGING_TYPE` is non-zero) use formatted strings, and the logging facility must be turned off for MISRA-C:2004 compliance. Even if enabled, the logging facilities use only the '`\n`' escape character which is defined in the ISO C standard.

Rule 4.2 (Required) Trigraphs shall not be used.

Compliance: YES

Notes:

Rule 5.1 (Required) Identifiers (internal and external) shall not rely on the significance of more than 31characters.

Compliance: YES

Notes:

Rule 5.2 (Required) Identifiers in an inner scope shall not use the same name as an identifier in an outer scope, and therefore hide that identifier.

Compliance: YES

Notes:

Rule 5.3 (Required) A `typedef` name shall be a unique identifier.

Compliance: YES

Notes:

Rule 5.4 (Required) A tag name shall be a unique identifier.

Compliance: YES

Notes: All tag names for data structure re-use the data structure identifier pre-pended with "_".

Rule 5.5 (Advisory) No object or function identifier with static storage duration should be reused.

Compliance: YES

Notes: All static storage in Abassi is global.

Rule 5.6 (Advisory) No identifier in one namespace should have the same spelling as an identifier in another namespace, with the exception of structure member and union member names.

Compliance: YES

Notes:

Rule 5.7 (Advisory) No identifier name should be reused.

Compliance: YES

Notes:

Rule 6.1 (Required) The plain `char` type shall be used only for the storage and use of character values.

Compliance: YES

Notes:

Rule 6.2 (Required) `signed` and `unsigned char` type shall be used only for the storage and use of numeric values.

Compliance: YES

Notes:

Rule 6.3 (Advisory) `typedefs` that indicate size and signedness should be used in place of the basic types.

Compliance: NO

Notes: As explained in Abassi User's Guide, by design Abassi only uses native "C" data type (except for `intptr_t`) for optimal real-time efficiency.

Rule 6.4 (Required) Bit fields shall only be defined to be of type unsigned `int` or `signed int`.

Compliance: YES

Notes: Abassi does not use bit fields.

Rule 6.5 (Required) Bit fields of signed type shall be at least 2 bits long.

Compliance: YES

Notes: Abassi does not use bit fields.

Rule 7.1 (Required) Octal constants (other than zero) and octal escape sequences shall not be used.

Compliance: YES

Notes: Octal constants are not used in Abassi.

Rule 8.1 (Required) Functions shall have prototype declarations and the prototype shall be visible at both the function definition and call.

Compliance: YES

Notes:

Rule 8.2 (Required) Whenever an object or function is declared or defined, its type shall be explicitly stated.

Compliance: YES

Notes:

Rule 8.3 (Required) For each function parameter, the type given in the declaration and definition shall be identical and the return types shall also be identical.

Compliance: YES

Notes:

Rule 8.4 (Required) If objects or functions are declared more than once, their types shall be compatible.

Compliance: YES

Notes:

Rule 8.5 (Required) There shall be no definitions of objects or functions in a header file.

Compliance: YES

Notes:

Rule 8.6 (Required) Functions shall be declared at file scope.

Compliance: YES

Notes:

Rule 8.7 (Required) Objects shall be defined at block scope if they are only accessed from within a single function.

Compliance: YES

Notes: Abassi's global variables are initialized in `OSstart()`. But when the compiler performs a zeroing of the uninitialized data section, the global variables that must be set to zero upon start of the RTOS are not initialized as the compiler does it at start-up. This means that some global variables are only accessed in a single function, which violates this rule. To fully abide to this rule when the port compiler zeroes the BSS, all there is to do is to set the token `OX_BSS_ZEROED` to a zero value. As a side effect, it will increase the code size of the RTOS.

Rule 8.8 (Required) An external object or function shall be declared in one and only one file.

Compliance: YES

Notes: Everything related to the RTOS is defined in `Abassi.h`

Rule 8.9 (Required) An identifier with external linkage shall have exactly one external definition.

Compliance: YES

Notes:

Rule 8.10 (Required) All declarations and definitions of objects or functions at file scope shall have internal linkage unless external linkage is required.

Compliance: YES

Notes:

Rule 8.11 (Required) The static storage class specifier shall be used in definitions and declarations of objects and functions that have internal linkage.

Compliance: YES

Notes:

Rule 8.12 (Required) When an array is declared with external linkage, its size shall be stated explicitly or defined implicitly by initialization.

Compliance: YES

Notes:

Rule 9.1 (Required) All automatic variables shall have been assigned a value before being used.

Compliance: YES

Notes:

Rule 9.2 (Required) Braces shall be used to indicate and match the structure in the non-zero initialization of arrays and structures.

Compliance: YES

Notes:

Rule 9.3 (Required) In an enumerator list, the "=" construct shall not be used to explicitly initialize members other than the first, unless all items are explicitly initialized.

Compliance: YES

Notes: Enumerations are not used in Abassi.

Rule 10.1 (Required) The value of an expression of integer type shall not be implicitly converted to a different underlying type if:

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.

Compliance: YES

Notes: Abassi does not comply with this rule when the build option `OS_SEARCH_ALGO` is set to a value greater than 1. As explained, no effort was made to make Abassi compliant when the build option `OS_SEARCH_ALGO` is set to a value greater than zero.

Rule 10.2 (Required) The value of an expression of floating type shall not be implicitly converted to a different underlying type if:

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..

Compliance: YES

Notes: Floating points numbers are not used in Abassi.

Rule 10.3 (Required) The value of a complex expression of integer type shall only be cast to a type that is not wider and of the same signedness as the underlying type of the expression.

Compliance: YES

Notes:

Rule 10.4 (Required) 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.

Compliance: YES

Notes: Floating points numbers are not used in Abassi.

Rule 10.5 (Required) If the bitwise operators ~ and << 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.

Compliance: YES

Notes:

Rule 10.6 (Required) A `U` suffix shall be applied to all constants of unsigned type.

Compliance: YES

Notes: Abassi does not comply with this rule when the build option `OS_SEARCH_ALGO` is set to a value greater than 1. As explained, no effort was made to make Abassi compliant when the build option `OS_SEARCH_ALGO` is set to a value greater than zero. As stated in the Abassi User Guide, the build option `OS_MAX_PEND_RQST` must be set to a numerical value with the suffix `U`.

Rule 11.1 (Required) Conversions shall not be performed between a pointer to a function and any type other than an integral type.

Compliance: YES

Notes:

Rule 11.2 (Required) 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.

Compliance: YES

Notes:

Rule 11.3 (Advisory) A cast should not be performed between a pointer type and an integral type.

Compliance: NO

Notes: The use of the data type `intptr_t` breaks this rule. But using `intptr_t` eliminates one argument to be passed to the kernel, and as the kernel is the function called for a large number of components, this decision helps reduce the CPU usage and the code space usage. This said, `intptr_t` is used exactly as it is intended to be. Unions, according to Rule 18.4, are not authorized, therefore Abassi does not create a union of integer and pointer, and instead uses `intptr_t`. As this rule is Advisory, and Rule 18.4 is Required, it was decided to sacrifice compliance of this rule instead.

Rule 11.4 (Advisory) A cast should not be performed between a pointer to object type and a different pointer to object type.

Compliance: YES

Notes:

Rule 11.5 (Required) A cast shall not be performed that removes any `const` or `volatile` qualification from the type addressed by a pointer.

Compliance: YES

Notes:

Rule 12.1 (Advisory) Limited dependence should be placed on the C operator precedence rules in expressions.

Compliance: YES

Notes: Parentheses are used everywhere in the code.

Rule 12.2 (Required) The value of an expression shall be the same under any order of evaluation that the standard permits.

Compliance: YES

Notes:

Rule 12.3 (Required) The `sizeof` operator shall not be used on expressions that contain side effects.

Compliance: YES

Notes:

Rule 12.4 (Required) The right-hand operand of a logical `&&` or `||` operator shall not contain side effects.

Compliance: NO

Notes: The components `TSKstate()`, `TSKisRdy()`, `TSKisBlk()` and `TSKisSusp()` cast to volatile entries in the task descriptors. It is necessary to cast as volatile the fields involved otherwise the optimizer could affect the result if any of these component is used in a loop, such as waiting for a task to become Ready.

Rule 12.5 (Required) The operands of a logical `&&` or `||` shall be *primary expressions*.

Compliance: YES

Notes:

Rule 12.6 (Advisory) 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 `?:`) .

Compliance: YES

Notes:

Rule 12.7 (Required) Bitwise operators shall not be applied to operands whose underlying type is signed.

Compliance: YES

Notes: Abassi does not comply with this rule when the build option `OS_SEARCH_ALGO` is set to a value greater than zero. As explained, no effort was made to make Abassi compliant when the build option `OS_SEARCH_ALGO` is set to a value greater than zero.

Rule 12.8 (Required) 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.

Compliance: YES

Notes:

Rule 12.9 (Required) The unary minus operator shall not be applied to an expression whose underlying type is unsigned.

Compliance: YES

Notes:

Rule 12.10 (Required) The comma operator shall not be used.

Compliance: NO

Notes: The component `EVTwait()`, a macro definition, uses the comma operator. This is the only component that does not comply with this rule. Implemention as a macro most of the time creates a smaller code footprint than what is required to call a function that would implement the same code. If compliance is required, the macro can be converted into a function, at the expense of code footprint.

Rule 12.11 (Advisory) Evaluation of constant unsigned integer expressions should not lead to wrap-around.

Compliance: YES

Notes:

Rule 12.12 (Required) The underlying bit representations of floating-point values shall not be used.

Compliance: YES

Notes: Floating point numbers are not used in Abassi.

Rule 12.13 (Advisory) The increment (++) and decrement (--) operators should not be mixed with other operators in an expression.

Compliance: NO

Notes: These are used in comparisons and resource distribution for optimal real time code generation.

Rule 13.1 (Required) Assignment operators shall not be used in expressions that yield a Boolean value.

Compliance: YES

Notes:

Rule 13.2 (Advisory) Tests of a value against zero should be made explicit, unless the operand is effectively Boolean.

Compliance: YES

Notes: Even with Booleans, all tests against zero are explicitly performed.

Rule 13.3 (Required) Floating-point expressions shall not be tested for equality or inequality.

Compliance: YES

Notes: Floating point numbers are not used in Abassi

Rule 13.4 (Required) The controlling expression of a `for` statement shall not contain any objects of floating type.

Compliance: YES

Notes: Floating point numbers are not used in Abassi

Rule 13.5 (Required) The three expressions of a `for` statement shall be concerned only with loop control.

Compliance: YES

Notes:

Rule 13.6 (Required) Numeric variables being used within a `for` loop for iteration counting shall not be modified in the body of the loop.

Compliance: YES

Notes:

Rule 13.7 (Required) Boolean operations whose results are invariant shall not be permitted.

Compliance: YES

Notes:

Rule 14.1 (Required) There shall be no unreachable code.

Compliance: YES

Notes:

Rule 14.2 (Required) All non-null statements shall either have at least one side effect however executed, or cause control flow to change.

Compliance: YES

Notes:

Rule 14.3 (Required) 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 whitespace character.

Compliance: YES

Notes:

Rule 14.4 (Required) The `goto` statement shall not be used.

Compliance: YES

Notes:

Rule 14.5 (Required) The `continue` statement shall not be used.

Compliance: YES

Notes:

Rule 14.6 (Required) For any iteration statement, there shall be at most one `break` statement used for loop termination.

Compliance: YES

Notes: `break` statements within loops are not used in Abassi.

Rule 14.7 (Required) A function shall have a single point of exit at the end of the function.

Compliance: YES

Notes: Enabling the logging facilities breaks this rule, and the logging facilities must be turned off for MISRA-C:2004 compliance

---

Rule 14.8 (Required) The statement forming the body of a `switch`, `while`, `do ... while`, or `for` statement shall be a compound statement.

Compliance: YES

Notes:

Rule 14.9 (Required) 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.

Compliance: YES

Notes:

Rule 14.10 (Required) All `if ... else if` constructs shall be terminated with an `else` clause.

Compliance: YES

Notes:

Rule 15.1 (Required) A `switch` label shall only be used when the most closely-enclosing compound statement is the body of a switch statement.

Compliance: YES

Notes: `switch` statements are only used in the logging facilities and the logging facilities must be turned off for MISRA-C:2004 compliance. Even with logging enabled, Abassi complies to this rule.

Rule 15.2 (Required) An unconditional `break` statement shall terminate every non-empty switch clause.

Compliance: YES

Notes: `switch` statements are only used in the logging facilities and the logging facilities must be turned off for MISRA-C:2004 compliance. Even with logging enabled, Abassi complies to this rule.

Rule 15.3 (Required) The final clause of a `switch` statement shall be the `default` clause.

Compliance: YES

Notes: `switch` statements are only used in the logging facilities and the logging facilities must be turned off for MISRA-C:2004 compliance.

Rule 15.4 (Required) A `switch` expression shall not represent a value that is effectively Boolean.

Compliance: YES

Notes: `switch` statements are only used in the logging facilities and the logging facilities must be turned off for MISRA-C:2004 compliance. Even with logging enabled, Abassi complies to this rule.

Rule 15.5 (Required) Every `switch` statement shall have at least one `case` clause.

Compliance: YES

Notes: `switch` statements are only used in the logging facilities and the logging facilities must be turned off for MISRA-C:2004 compliance. Even with logging enabled, Abassi complies to this rule.

Rule 16.1 (Required) Functions shall not be defined with a variable number of arguments.

Compliance: YES

Notes: `printf` are only used in the logging facilities and the logging facilities must be turned off for MISRA-C:2004 compliance.

Rule 16.2 (Required) Functions shall not call themselves, either directly or indirectly.

Compliance: YES

Notes:

Rule 16.3 (Required) Identifiers shall be given for all of the parameters in a function prototype declaration.

Compliance: YES

Notes:

Rule 16.4 (Required) The identifiers used in the declaration and definition of a function shall be identical.

Compliance: YES

Notes:

Rule 16.5 (Required) Functions with no parameters shall be declared and defined with the parameter list void.

Compliance: YES

Notes:

Rule 16.6 (Required) The number of arguments passed to a function shall match the number of parameters.

Compliance: YES

Notes: It may look like this rule is not respected in the assembly code when the kernel is entered, but all arguments are passed to `Abassi()`, except that only the argument `Ops` is set to a 0, and with `Ops` set to 0, the two others arguments are not used by `Abassi()`.

Rule 16.7 (Advisory) 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.

Compliance: YES

Notes:

Rule 16.8 (Required) All exit paths from a function with non-void return type shall have an explicit return statement with an expression.

Compliance: YES

Notes:

Rule 16.9 (Required) A function identifier shall only be used with either a preceding `&`, or with a parenthesized parameter list, which may be empty.

Compliance: YES

Notes:

Rule 16.10 (Required) If a function returns error information, then that error information shall be tested.

Compliance: YES

Notes:

Rule 17.1 (Required) Pointer arithmetic shall only be applied to pointers that address an array or array element.

Compliance: YES

Notes:

Rule 17.2 (Required) Pointer subtraction shall only be applied to pointers that address elements of the same array.

Compliance: YES

Notes:

Rule 17.3 (Required) >, >=, <, <= shall not be applied to pointer types except where they point to the same array.

Compliance: YES

Notes:

Rule 17.4 (Required) Array indexing shall be the only allowed form of pointer arithmetic.

Compliance: YES

Notes: Abassi complies to this rules as long as the mailboxes are not part of the build (the build option `OS_MAILBOX`). If the mailboxes are part of the build, because the mailboxes have a selectable buffer size, then a pointer is used to hold the base address of the buffer, and this buffer is accessed using read and write indexes.

Rule 17.5 (Advisory) The declaration of objects should contain no more than two levels of pointer indirection.

Compliance: YES

Notes:

Rule 17.6 (Required) 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.

Compliance: YES

Notes: It may look like the kernel breaks this rule when it queues the requests when in an interrupt context, but Abassi does not allow the destruction of descriptors. Therefore all objects memorized when queuing requests are permanent.

Rule 18.1 (Required) All structure and union types shall be complete at the end of the translation unit.

Compliance: YES

Notes:

Rule 18.2 (Required) An object shall not be assigned to an overlapping object.

Compliance: YES

Notes:

Rule 18.3 (Required) An area of memory shall not be used for unrelated purposes.

Compliance: YES

Notes:

Rule 18.4 (Required) Unions shall not be used.

Compliance: YES

Notes:

Rule 19.1 (Advisory) `#include` statements in a file should only be preceded by other preprocessor directives or comments.

Compliance: YES

Notes: For most ports Abassi complies with this rule, but a few ports need special include files for start-up configuration and / or for added functionality. As all port specific code is grouped together in `Abassi.h`, these `#include` are not at the beginning of the file.

Rule 19.2 (Advisory) Non-standard characters should not occur in header file names in `#include` directives.

Compliance: YES

Notes:

Rule 19.3 (Required) The `#include` directive shall be followed by either a `<filename>` or "`filename`" sequence.

Compliance: YES

Notes:

Rule 19.4 (Required) C macros shall only expand to a braced initializer, a constant, a string literal, a parenthesized expression, a type qualifier, a storage class specifier, or a do-while-zero construct.

Compliance: YES

Notes: 1) Only if the build option `OS_RUNTIME` is set to a negative value, meaning compile time creation of descriptors is disabled. This means for MISRA-C:2004 compliance the components `TSK_STATIC()`, `SEM_STATIC()`, `MTX_STATIC()`, `MBX_STATIC()` and `TIM_STATIC()` cannot be used, nor be available.

2) For most ports Abassi complies with this rule, but a few ports need extra functions for added functionality. As all port specific code is grouped together in `Abassi.h`, and not in `Abassi.c`, extra functions are defined by a pre-processor macro. Doing so keeps the kernel code untouched when a new port is added. If compliance is required, the culprit macro can be manually expanded in `Abassi.c`.

Rule 19.5 (Required) Macros shall not be `#define'd` or `#undef'd` within a block.

Compliance: YES

Notes:

Rule 19.6 (Required) `#undef` shall not be used.

Compliance: YES

Notes:

Rule 19.7 (Advisory) A function should be used in preference to a function-like macro.

Compliance: NO

Notes: Many components are macros.

Rule 19.8 (Required) A function-like macro shall not be invoked without all of its arguments.

Compliance: YES

Notes:

Rule 19.9 (Required) Arguments to a function-like macro shall not contain tokens that look like preprocessing directives.

Compliance: YES

Notes:

Rule 19.10 (Required) 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 `##`.

Compliance: YES

Notes:

Rule 19.11 (Required) All macro identifiers in preprocessor directives shall be defined before use, except in `#ifdef` and `#ifndef` preprocessor directives and the `defined()` operator.

Compliance: YES

Notes:

Rule 19.12 (Required) There shall be at most one occurrence of the `#` or `##` preprocessor operators in a single macro definition.

Compliance: YES

Notes: Only if the build option `OS_RUNTIME` is set to a negative value, meaning compile time creation of descriptors is disabled. This means for MISRA-C:2004 compliance, the components `TSK_STATIC()`, `SEM_STATIC()`, `MTX_STATIC()`, `MBX_STATIC()` and `TIM_STATIC()` cannot be used, nor be available.

Rule 19.13 (Advisory) The `#` and `##` preprocessor operators should not be used.

Compliance: YES

Notes: Only if the build option `OS_RUNTIME` is set to a negative value, meaning compile time creation of descriptors is disabled. This means for MISRA-C:2004 compliance, the components `TSK_STATIC()`, `SEM_STATIC()`, `MTX_STATIC()`, `MBX_STATIC()` and `TIM_STATIC()` cannot be used, nor be available.

Rule 19.14 (Required) The `defined` preprocessor operator shall only be used in one of the two standard forms.

Compliance: YES

Notes:

Rule 19.15 (Required) Precautions shall be taken in order to prevent the contents of a header file being included twice.

Compliance: YES

Notes:

Rule 19.16 (Required) Preprocessing directives shall be syntactically meaningful even when excluded by the preprocessor.

Compliance: YES

Notes:

Rule 19.17 (Required) All `#else`, `#elif`, and `#endif` preprocessor directives shall reside in the same file as the `#if` or `#ifdef` directive to which they are related.

Compliance: YES

Notes:

Rule 20.1 (Required) Reserved identifiers, macros, and functions in the standard library shall not be defined, redefined, or undefined.

Compliance: YES

Notes:

Rule 20.2 (Required) The names of Standard Library macros, objects, and functions shall not be reused.

Compliance: YES

Notes:

Rule 20.3 (Required) The validity of values passed to library functions shall be checked.

Compliance: YES

Notes:

Rule 20.4 (Required) Dynamic heap memory allocation shall not be used.

Compliance: YES

Notes: Only if the build options `OS_STATIC_XXX` are non-zero. These build options define how many tasks, semaphore, mailboxes, etc. the application needs. If any of these build option is zero, and a descriptor is needed by the application, dynamic memory allocation will be used, either from the memory pool defined by `OS_ALLOC_SIZE` or through `malloc()` if `OS_ALLOC_SIZE` is set to zero.

Rule 20.5 (Required) The error indicator `errno` shall not be used.

Compliance: YES

Notes:

Rule 20.6 (Required) The macro `offsetof` in the `stddef.h` library shall not be used.

Compliance: YES

Notes:

Rule 20.7 (Required) The `setjmp` macro and the `longjmp` function shall not be used.

Compliance: YES

Notes:

Rule 20.8 (Required) The signal handling facilities of `signal.h` shall not be used.

Compliance: YES

Notes:

Rule 20.9 (Required) The input/output library `stdio.h` shall not be used in production code.

Compliance: YES

Notes: Only the logging facilities (when the build option `OS_LOGGING_TYPE` is non-zero) needs `stdio.h`, and the logging facility must be turned off for MISRA-C:2004 compliance.

Rule 20.10 (Required) The functions `atof`, `atoi`, and `atol` from the library `stdlib.h` shall not be used.

Compliance: YES

Notes: Floating points numbers are not used in Abassi.

Rule 20.11 (Required) The functions `abort`, `exit`, `getenv`, and system from the library `stdlib.h` shall not be used.

Compliance: YES

Notes:

Rule 20.12 (Required) The time handling functions of `time.h` shall not be used.

Compliance: YES

Notes:

Rule 21.1 (Required) Minimization of runtime failures shall be ensured by the use of at least one of:

a. static analysis tools/techniques

b. dynamic analysis tools/techniques

c. explicit coding of checks to handle runtime faults.

Compliance: YES

Notes: There are no run-time checks performed in Abassi.

The two most common cases of run-time failures are:

- Task stack overflow
- Not enough descriptors in a class are reserved with the build option `OS_STATIC_XXX`. When a class of descriptors are not reserved with `OS_STATIC_XXX`, then a dynamic memory allocation failure could happen.

These run time failures can easily be controlled during the design phase of the application.