FORD LDM Localization - 功能 #3354

Task # 3240 (进行中): FORD文档输出

功能 # 3349 (进行中): SWQA文档

TDR_RQT_003701_023449 Compliance to Coding Standards

2025-03-04 09:50 - 稚媛 黄

状态:	进行中	**开始日期:**	2025-03-04
优先级:	普通	**计划完成日期:**	2025-03-31
指派给:	槐 杨	**% 完成:**	50%
类别:		**预期时间:**	0.00 小时
目标版本:	FORD_TDR_Documents	**耗时:**	0.00 小时

描述

1、C语言：必须符合签署供应商合作协议（SOBA）时最新的MISRA C编码标准，包括所有规则、指令和修正。
2、C++语言：必须符合签署SOBA时最新的AUTOSAR C++编码标准，包括所有规则、指令和修正。（注：AUTOSAR C++ 14直接替代了MISRA C++ 2008标准。）
3、C/C++语言：必须符合福特编码标准。
1）对于未命名的数值常数，避免使用未命名的数字常量，优先使用有意义的命名常量。
可以接受的用法：当定义和可用时，使用True和False，而不是文字1和0。
不可接受的用法：不建议将字符串文字作为代码的一部分。
2）针对于包含文件，避免多重声明，避免别名和悬空引用。
可接受的使用：防止在单个编译单元中多次处理同一个头文件，例如，使用#ifndef、#define和#endif来保护头文件，条件编译会增加调试和管理不同代码变体的难度，在生成代码变体时，优先在构建级别包含或排除文件，而不是在源文件中使用条件编译。
不推荐的使用：不要#include一个.c文件，不要在头文件中放置可执行代码。
3） ~~当多个任务同时访问一个资源时，资源或其提供的数据可能会被破坏。~~
~~例如，一个任务在读取NVM（非易失性存储器）之前，必须等待所有其他NVM访问完成。~~
~~并发和共享资源问题是导致保修问题的主要原因之一，仅次于“编程疏忽”。~~
~~共享资源的示例：~~
~~MCU内部的资源（如变量、NVM、寄存器、硬件I/O等），由单核上的多个任务访问，单核上的任务和ISR（中断服务例程）访问的内部资源，多核上的任务和/或ISR访问的内部资源，多个ECU访问的外部资源。~~
~~资源保护的示例：~~
~~互斥锁和信号量，获取操作系统的“资源”，任务间变量，独占区域~~
~~可接受的使用：~~
~~保护共享资源：~~
~~任何共享变量、寄存器、I/O电路或辅助函数都应通过管理器、操作系统资源（如信号量）或任务间变量进行保护。~~
~~ISR中的资源保护：~~
~~所有在ISR内部访问的资源都应受到保护，因为上下文增加了资源安全使用的复杂性。~~
~~外部设备的保护：~~
~~所有与微控制器共享连接的外部设备都应受到保护。~~
~~确保这些外部设备遵循HSIS（硬件软件接口规范）中规定的限制、建议和访问控制。~~
~~不推荐的使用：~~
~~不要创建自己的管理器来保护共享资源，除非操作系统或开发框架未提供保护机制。~~
——本模块选用MCU为单核，不存在同时操作的情况
4)对于值比较的情况
以下情况不需要使用不等式：
变量声明为整数但用作枚举。
测试整数是否等于位表示的最小值或最大值。
测试整数是否为零。
switch语句中的default情况可以处理所有超出范围的条件。
不推荐的使用：
~~避免隐式比较：~~
~~例如：if (take_branch)，应明确写出条件，如 if (take_branch == true)。~~
~~避免使用“双重否定”进行比较：~~
~~例如：if (NOT_DISABLED != false)，应直接写成 if (NOT_DISABLED)。~~
~~类似地，#if (NOT_DISABLED != 0) 应简化为 #if (NOT_DISABLED)。避免使用浮点值作为循环终止条件或循环索引：~~
~~浮点值的精度问题可能导致不可预测的行为。~~
——不影响代码执行
5）使用条件语句时，最好使用简单、易于理解的表达方式，保证每行一个语句
不推荐使用复合语句，避免在测试中产生副作用，在if()或while()语句中避免嵌入赋值或其他可能产生副作用的操作
6）模块的设计意图行为需要在系统和ECU级别明确定义。ECU还需要一种机制来防止代码或硬件的未定义行为。
可接受的使用：
处理“除以零”问题：

检测并返回错误代码给调用函数。
检测并提供默认的分母以避免错误。
调用函数在收到此类错误代码时，可以执行备用行为。
使用运行时分析工具：
检测缓冲区溢出或下溢、内存泄漏、通过过期指针访问“ 僵尸 ”内存区域等问题。
例如：使用Valgrind、Polyspace Code Prover等工具。
确保设计文档准确描述所有公共和全局成员及函数参数：
确保使用符合设计意图。
正确管理共享数据：
~~当数据在多个任务之间或任务与ISR之间共享时，确保数据的正确管理。~~
不推荐的使用：
避免在应用代码中使用#pragma：
#pragma通常与编译器相关，可能降低代码的可移植性。
如果必须使用#pragma：在项目的软件开发计划中记录使用条件,通过审查或工具确认其合规性。
7)在开发周期中，应审查并采用这些规则。静态分析工具无法覆盖所有未定义行为，因此项目负责人需要手动审查附录H中无法静态检查的规则:
可接受的使用：
启用所有编译器警告：
在构建过程中启用所有编译器警告，以捕捉潜在问题。
使用足够的括号：
使用括号明确计算顺序，提高代码可读性。
汇编代码注释：
在使用汇编代码时，确保为每条汇编指令提供足够的注释。
内存处理函数的使用规则：
目标缓冲区必须至少与size/count参数一样大。
所有缓冲区参数必须为非NULL。
对于memmove和memcpy：
源缓冲区必须至少与size/count参数一样大。
源缓冲区和目标缓冲区不能重叠。
使用静态分析工具：
在项目级别始终使用静态分析工具，并修复发现的问题。
不推荐的使用：
不要使用系统默认值：
例如，堆栈、堆栈大小、MCU寄存器等的值不应使用系统默认值，而应显式配置
8)符号作用域,C语言允许创建对对象的多个引用，但不强制这些引用的一致性。允许别名的生命周期超过被引用对象的生命周期可能导致未定义行为，并引发安全或安全问题。通过不同类型的别名引用对象可能导致数据损坏和不安全行为。
接受的使用：
限制别名的创建：
仅在绝对必要时创建对象的别名。
控制别名的生命周期：
防止对已超出作用域的对象的“ 僵尸 ”引用。
确保别名类型兼容：
例如，int类型和指向int的指针类型是兼容的
不推荐的使用：
确保通过解引用别名读取的数据是有效的：
避免访问无效或已释放的内存。
创建别名时，不要使引用对象的类型与被引用对象的类型不同：
例如，不要将int类型的对象别名定义为float类型。
不要返回函数局部变量的地址：
局部变量在函数返回后会被销毁，返回其地址会导致未定义行为。
9）字符串处理
可接受的使用：
字符串应定义为char类型的数组：
不要将字符串定义为signed char或unsigned char类型。
使用安全的字符串处理函数：
优先使用C11标准中提供的边界检查函数（如strcpy_s、strcat_s、scanf_s、sprintf_s、gets_s等）。
允许使用的标准库函数（需遵守额外规则）：strncpy、strncpy_s、strncat、strncat_s。
使用规则：
目标缓冲区必须足够大以容纳源数据。源缓冲区必须至少与size/count参数一样大。
源缓冲区和目标缓冲区不能重叠。两个缓冲区必须为char类型的数组。
两个缓冲区必须以空字符（\0）结尾。size/count参数必须为size_t或rsize_t类型，必须非零，必须适当限制，且必须小于等于RSIZE_MAX。
不推荐的使用：
避免使用不安全的字符串处理函数：例如：strcpy、strcat、scanf、sprintf、gets等，这些函数容易导致缓冲区溢出和其他安全问题。
避免使用strdup：strdup是POSIX函数，不是ANSI/ISO C标准的一部分，且会分配内存，可能导致内存管理问题
10）严格验证文件的来源和内容，防止恶意或损坏的文件影响系统
仅处理已知和受信任的文件类型

处理未知数据值时采取防御性措施
11）“ 输入数据 ”不仅限于函数参数，任何由函数读取的内存或外设数据都被视为该函数的输入。
规则：
所有输入数据必须由使用它的函数进行验证：
验证可能包括检查有效数据范围或特定值。
尽可能对来自模块外部的所有输入进行有效性、适当性和发送者身份验证。
假设所有输入数据不安全，直到检查或验证：
对于编译时已知的数据，可以使用编译时断言来验证，以减少运行时开销。
涉及缓冲区或数组时，必须进行特定检查以确保所有访问都在数组边界内：
调用函数必须确保地址和大小参数有效。
尽可能在影响任何输出之前验证所有输入数据：
例如，如果未激活适当的诊断会话，则阻止诊断操作。
发现无效数据时，必须实施适当的反应：例如，返回错误指示、提供合理的默认值、记录发现等。
12）“ 输出数据 ”不仅限于函数参数，任何由被调用函数写入的内存或外设数据都被视为该函数的输出，即使写入的值未更改。
规则：
调用函数必须检查被调用函数的所有输出数据的“ 合理性 ”：
验证可能包括检查有效数据范围或特定值，但不限于此。
当函数的输出涉及写入缓冲区或数组时，必须进行特定检查以确保所有写入都在缓冲区边界内：
调用函数必须确保地址和大小参数有效。
调用函数必须假设所有输出数据无效，直到检查：
如果被调用函数直接返回错误代码，除非设计保证输出值，否则所有其他输出数据应视为可疑（即状态未知）。
“ 合理性检查 ”失败时，必须实施适当的反应：
例如，传递错误指示、用合理的默认值替换错误值、记录日志等。
对于任何返回值的函数，必须检查所有可能的返回值并采取适当的反应：
此规则适用于所有返回类型非void的函数，即使是C标准库中的函数。
如果返回值不重要，必须显式将其强制转换为void。

4、提供经批准的工具进行静态分析、复杂度指标收集和MISRA-C/AUTOSAR C++合规性检查

历史记录

#1 - 2025-03-12 14:44 - 稚媛 黄

- 描述 已更新。

- 状态 从 新建 变更为 进行中

#2 - 2025-03-12 14:51 - 稚媛 黄

- 描述 已更新。

#3 - 2025-03-19 10:33 - 稚媛 黄

- 描述 已更新。

#4 - 2025-03-19 10:36 - 稚媛 黄

- 描述 已更新。

#5 - 2025-03-19 10:51 - 稚媛 黄

- 描述 已更新。

#6 - 2025-03-19 13:10 - 稚媛 黄

- 描述 已更新。

#7 - 2025-03-19 14:20 - 稚媛 黄

- 描述 已更新。

#8 - 2025-03-19 14:23 - 稚媛 黄

- 描述 已更新。

#9 - 2025-03-19 14:52 - 稚媛 黄

- 描述 已更新。

#10 - 2025-03-19 15:29 - 稚媛 黄

- 描述 已更新。

#11 - 2025-03-25 13:09 - 涛 陆

- 描述 已更新。

- 指派给 从 稚媛 黄 变更为 槐 杨

- % 完成 从 0 变更为 50

根据需求，进行静态测试规则编写