MM32 IEC 60730 ClassB For M0 软件库

用户手册

V0.1

目录

前言

本工作指导认证评审人员在 DUT（测试设备）上确定具体安全机制的实施。

安全技术在电子应用中的作用越来越重要。对零部件的安全要求水平正在稳步提高，电子设备的制造商在其设计中包含了许多新的技术解决方案。提高安全性的软件技术正在不断发展。有关硬件和软件安全要求的标准也在不断发展中。

关键的 IEC 标准是 IEC 60730-1 和 IEC 60335-1，从第 4 版开始与 UL/CSA 60730-1 和 UL/CSA 60335-1 很好地协调(以前的 UL/CSA 版本还引用了 UL1998 规范)。它们涵盖了家用电子设备在家庭和类似环境中的安全与保障。

装有电子电路的器具必须进行元件失效试验。这里的基本原则是，在任何部件发生故障的情况下，设备必须保持安全。从这个角度来看，微控制器是一个电子元件，就像任何其他电子元件一样。如果安全依赖于一个电子元件，它必须在连续两次故障后保持安全。这意味着设备必须在一个硬件故障和微控制器不操作(复位或不正常操作)的情况下保持安全。

所需条件在 IEC 60335-1 标准的附录 Q 和 R 以及 IEC 60730-1 标准的附录 H 中有详细的定义。60730-1 标准定义了三类：

- **A 类**: 安全不依赖于软件。
- **B 类**: 软件防止不安全操作。
- **C 类**: 软件用于防止特殊危害。

1 概览

本手册详细介绍了基本结构，其中针对 B 类需求的自测程序和方法收集在公共 STL 库和特定于产品的 STL 库目录下。其余的驱动程序大多是特定于应用程序的，并且在最终客户项目中可能会根据用户应用程序进行更改或替换。

表 1 总结了这些项目的详细结构以及包含在 STL 堆栈的公共部分和特定部分中的文件列表。

表 1. 通用 STL 包的结构

STL	通用的 STL 栈源文件	
	文件	说明
启动测试	IEC60730_B_startup.c	启动 STL 流程控制& 启动时检测
	IEC60730_B_cpustartIAR.s	IAR 环境下 CPU 启动时自检
	IEC60730_B_cpustartKEIL.s	MDK 环境下 CPU 启动时自检
运行时测试	IEC60730_B_runtimetest.c	运行时自检结构
	IEC60730_B_flashtest.c	启动和运行时的 FLASH 检测
	IEC60730_B_transpRam.c	部分 RAM 检测
	IEC60730_B_clocktest.c	时钟检测
	IEC60730_B_aux.c	包含了错误处理函数、看门狗检测以及初始化函数、CSS 中断处理、SysTick 时基处理和运行时 RAM 检测的调用接口
头文件	IEC60730_B_clock.h	时钟检测头文件
	IEC60730_B_cpu.h	CPU 检测头文件
	IEC60730_B_crc32.h	Flash 检测头文件
	IEC60730_B_init.h	运行时检测的初始化函数接口
	IEC60730_B_lib.h	整体的 STL 库的头文件包含管理
	IEC60730_B_param.h	产品特定 STL 配置文件
	IEC60730_B_Ram.h	RAM 检测头文件
	IEC60730_B_startup.h	启动时检测所需头文件
	IEC60730_B_var.h	B 类变量定义

60335-1 标准要求根据 60730-1 标准表 H.11.12.7，结合软件措施来控制表 R.1 和 R.2 中规定的故障/错误条件 ：

表 R.1 总结了可与表 H.11.12.7 中给出的 B 级要求相符的一般条件。

表 R.2 总结了可与 60730-1 标准 C 级要求相符的特定条件，用于解决特定危险的特定结构。

同样，如果软件仅用于功能目的，则 R.1 和 R.2 要求不适用。

本申请说明和相关的 IEC60730_B 软件包的范围为 60730-1 标准和相应条件下的 B 类规范，总结在 60335-1 标准的表 R.1 中。

如果功能安全依赖于 B 类软件，如果设备发生其他故障，代码必须防止危险。自检软件能检测故障并做相应处理。在安全关键程序中发生的意外软件故障不一定会由于另一个应用的冗余软

件程序或硬件保护功能而造成危险。这不是更严重的 C 级级别的情况，即安全关键软件的故障会由于缺乏下一等级的保护机制而导致危险。

2 STL 固件软件包中包含的通用测试

经过认证的 MindMotion STL 固件包由以下微特定软件模块组成：

- CPU 寄存器测试
- 系统时钟监控
- RAM 功能检查
- Flash 闪存 CRC 完整性检查
- 看门狗自检
- 堆栈上溢监控

表 2给出了用于 MCU 特定测试的方法的概述（在以下章节中详细描述）。

用户可以将部分或全部经过认证的软件模块纳入其项目中。如果它们没有被更改，并根据这些指导方针进行集成，那么获得经过认证的最终应用程序所需的时间和成本将大大减少。

当测试被删除时，用户必须考虑副作用，因为未应用的测试也可以在其他组件的测试中发挥作用。

表 2. STL 包相关模块的测试方法

待验证的表 R.1 中的组件	使用的方法	参考 IEC 60335-1 附录 R 和 IEC 60730-1 附录 H	
		组件	IEC 60730 参考
CPU 寄存器	所有寄存器和标记的功能测试均在启动时完成，包括 R13（栈指针）、R14（链接寄存器）和 PSP（进程栈指针）。运行时的测试不 检测 R13，R14，PSP 和标志。测试栈指针是否上溢，（下溢通过非 直接方法检查）链接寄存器由 PC 监控进行测试。如果发现任何错误，软件直接跳至故障保护程序。	1.1	H.2.16.5 H.2.16.6 H.2.19.6
程序计数器	当程序计数器丢失或终止时，两个不同的看门狗可复位设备，它们在两个独立的时钟源运行。窗口看门狗由主振荡器驱动，可执行时 隙监控，独立看门狗由内部低速 RC 振荡器驱动，一旦启用将无法禁用。程序控制流程可采用专用软件方法监控。	1.3	H.2.18.10.2 H.2.18.10.4

寻址和数据路径	并非所有的产品都满足本次测试的公认方法。这种缺陷可以通过实现一组更广泛的间接方法来弥补，如 RAM 功能和闪存完整性测试、程序定时和流控制、B 类变量完整性和其他硬件方法支持的堆栈边界检查，如正确处理 CPU 异常。	4.3, 5.1 and 5.2	H.2.19.18.1 H.2.19.18.2 （间接测试）
时钟	采用两个独立时钟源交叉检查来进行测量。被测量的频率控制定时器时钟，另一个来给定定时器时钟输入。例如，使用内部低速 RC 振 荡器时基可检测外部晶振（谐波/次谐波）的错误频率。	3	H.2.18.10.1 H.2.18.10.4
常量内存	完整内存的 32 位 CRC 校验和测试在启动时完成，部分内存测试在运行时间重复进行（逐块）。使用快速内置 32 位 CRC 计算单元。	4.1	H.2.19.4.1 H.2.19.8.1
变量内存	March C-完整内存测试在启动时完成，部分内存测试在运行时重复（在 B 类存储区域逐块进行）。在耦合故障优化覆盖的测试中，遵循 RAM 内物理地址的扰乱次序。Faster March X 可用于在运行时间进行测试。可采用双重冗余字保护（在非邻近内存空间保存相反值）的方法对安全相关的 B 类变量进行保护，运行期间不对 A 类变量空间、栈及未使用空间进行测试。	4.2	H.2.19.6.2 H.2.19.8.2

3 软件包

本节重点介绍了 MindMotion 软件解决方案中使用的基本通用原则。本文描述了工作区组织及其配置和调试功能。受支持的开发环境之间的差异。代码库集成可基于“B 类软件库的移植（第 7 章）”进行。

3.1 故障安全模式

当自检过程检测到故障时，将调用专用处理函数 FailSafePOR（）。该例程在 IEC60730_B_aux.c 文件中有预定义。这个过程的目标是提供一个唯一的输出，并允许用户立即作出反应。

3.2 安全相关变量及堆栈边界控制

强烈建议以特定的方式处理与系统安全相关的临界值。每个 B 类变量作为一对两个互补的值存储在两个单独的 RAM 区域中。正常的和冗余的互补值总是被放置在非相邻的内存位置。运行时通过特定的中断子程序逐步对这些 RAM 区域执行 March C-或 March X 测试。用于临时存储和反向恢复测试区域的缓冲区也在永久测试的范围内。
在使用该值之前，用户必须确保总是对每对数据进行完整性比较。如果任何数据对的完整性已损坏，则必须调用故障安全模式。如果有意更改变量的值，则需要更新两个存储位置，以保持存储对的正确完整性。

图 1 显示了一个 RAM 配置的示例。

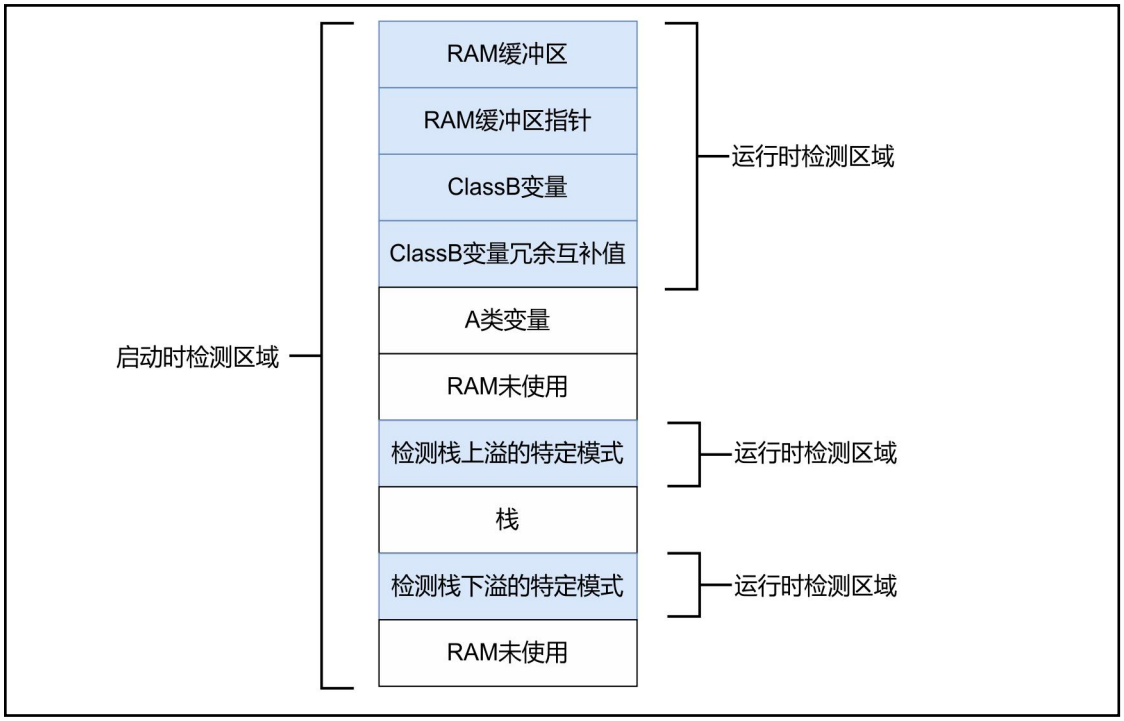

图 1. RAM内存配置的示例

用户必须对齐测试区域的大小，以增加单个透明测试步骤，同时考虑测试的第一步和最后一步RAM测试区域的大小。

这就是为什么用户必须在专门用于B类变量的区域的开始和结束时填充间隙。这些间隙的大小必须与测试单块的重叠覆盖区相对应。

备份缓冲区和指向B类区域的指针必须分配到专用于B类变量的区域之外，在每次整个B类区域测试周期完成时分别测试的特定位置。

当应用程序使用多个堆栈区域时，建议按边界模式的特定数据分隔相邻区域，以检查所有指针是否只在其专用区域内操作。

在运行时，透明RAM测试不会对堆栈和非安全RAM区域进行检查。

3.3 程序流程控制

程序流程控制是标准强烈推荐的方法，这是因为它能有效确保所有特定代码能正确执行并通过。该检查采用特定的软件方法。为识别代码流程中的所有关键点（组件测试块），应定义独特的标签（常量数），从而确保不跳过任何块，且所有流程均如期执行。独特的标签在两个互补计数器中处理，符合 B 类变量标准。主要原理是在每次处理任何重要的测试块时，这两个互补计数器的值（增加或减去独特的标签值）发生对称的四步变化。其中两个检查步骤安排在调用程序（主流程）层的调用块外。这确保了块从主流程层级被成功调用（在调用前及从调用程序返回后处理）。下两个步骤是在调用程序内执行，以确保块正确完成（在进入程序后及从程序返回前处理）。

按控制流程顺序调用执行组件测试的例程，显示四步检查服务，图 2 给出了示例。因为所有这些点都通过计算一组互补计数对而检查，所以该方法可降低 CPU 的负载。由于有些调用/返回编号及进入/退出点始终相同，在每个块完全通过后，储存在计数对内的值必须始终互补。检查计数对完整性时，对几个执行流程检查点进行评估并放入代码流程。如果任何检查点的计数器不互补或其不包含预期值，应调用故障保护例程。如*图 2*所示。

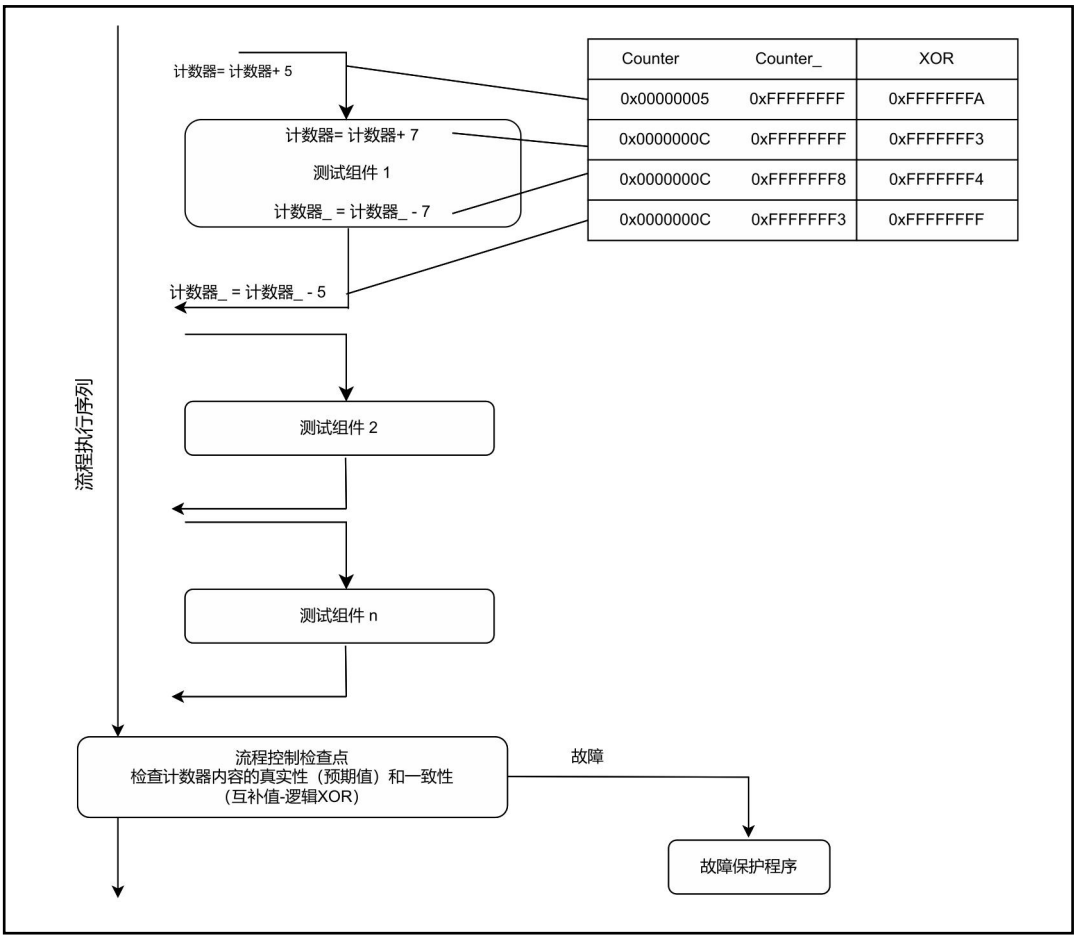

图 2. 控制流程四步检查原理

3.4 定义新的安全变量和被检查的内存区域

为确保变量内存中安全临界数据（B 类）的冗余，需在 CLASS_B_RAM 和 CLASS_B_RAM_REV 中进行双重存储。所有其他定义变量（无任何特殊属性）均被认为是 A 类变量，在透明 RAM 测试过程中不受检查。

3.5 软件包的配置和调试

IEC60730_B 包必须根据实际产品的情况配置正确的设置。

有时，一个简单的应用程序结构涉及挂起或排除 STL 包的一个功能部分（例如，系统从内部时钟输入）。相反，一些特性可以在包调试期间临时添加，因为它们有助于此阶段的开发人员。本节介绍如何配置、修改和调试该解决方案。

根据最终应用程序，可以跳过一些运行时测试。如果测试的周期性与使用频率一致，那么通电测试就足够了，可以避免透明/运行时测试（例如，洗衣机的情况：用户每次使用时打开/关闭应用程序）。这一点必须与选定的测试机构就逐个案例进行讨论。

建议在测试结束阶段执行窗口功能，在调试时冻结看门狗选项。

用户必须遵循初始 RAM 和 Flash 测试的时间，尤其当测试区域较大且整个看门狗周期不够用时。在这种情况下，测试必须分为几个部分，必须完成在它们之间额外进行看门狗刷新服务。这些服务无需是代码中任何循环的组成部分。

根据 60370-1 标准，栈上溢检测和看门狗自检不是强制性的。他们可在用于间接测试某些功能时添加，如果用户喜欢使用其他方法，也可将其禁用或跳过。

4 软件库结构（程序顺序）

B 类例程被分为两个主要过程，即启动和定期运行时自检。在应用前，必须通过设置块来初始化定期运行时测试。所有过程均在有效的调用-被调用控制流程中。

所有的 B 类变量都具有冗余特性，在用户指定的 B 类变量空间内做了双重存储。作为运行时测试的一部分，该空间被分为两个单独的 RAM 区，始终进行检测。

图 3 显示 B 类软件包集成入用户软件解决方案的基本原理。

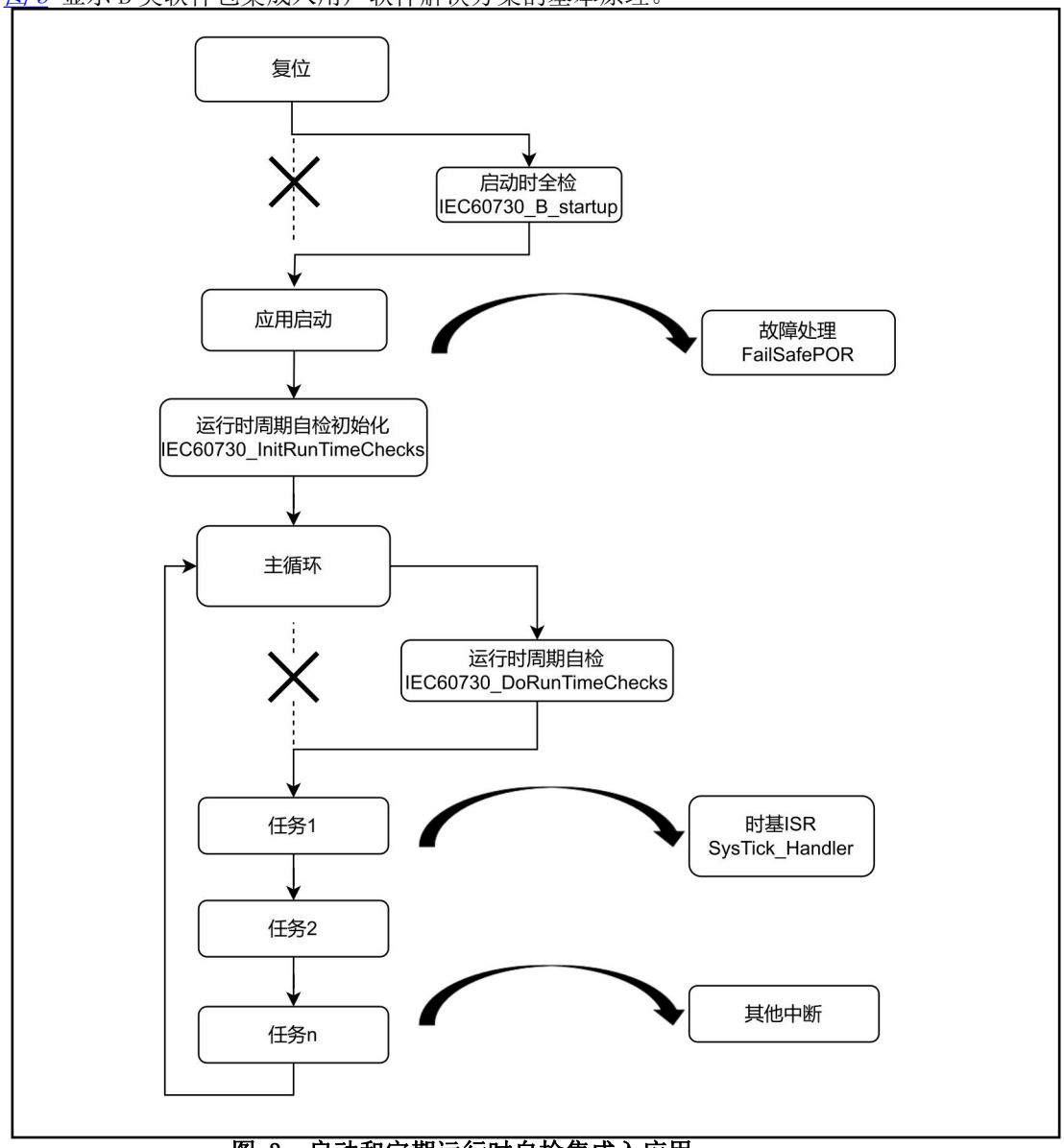

图 3. 启动和定期运行时自检集成入应用

原则上，当 IEC60730_B 模块集成到一个应用程序中时，用户必须提供以下步骤：

- 在用户程序启动之前执行初始启动测试
- 定期执行专门时间段（与安全时间相关）设置的运行时自检
- 应用运行时，设置独立和窗口看门狗，以防止其到期（理想的情况是通过 IEC60730_B 测试周期对其刷新进行微调）
- 为 RAM 和 Flash 的启动和运行时测试设置正确的测试区
- 注意测试的错误结果，并进行合适的操作来保证系统的安全
- 处理安全状态程序及其内容
- 处理 HardFault 异常程序及其内容
- 防止可能与应用 SW 发生的冲突
- 运行与应用程序安全任务相关的特定微控制器部件的测试
- 排除所有与任何安全任务不相关的调试功能，仅将其用于调试或测试

4.1 启动自检

复位微控制器后，执行首次检查时，应在初始化阶段运行启动自检，如图 4所示。

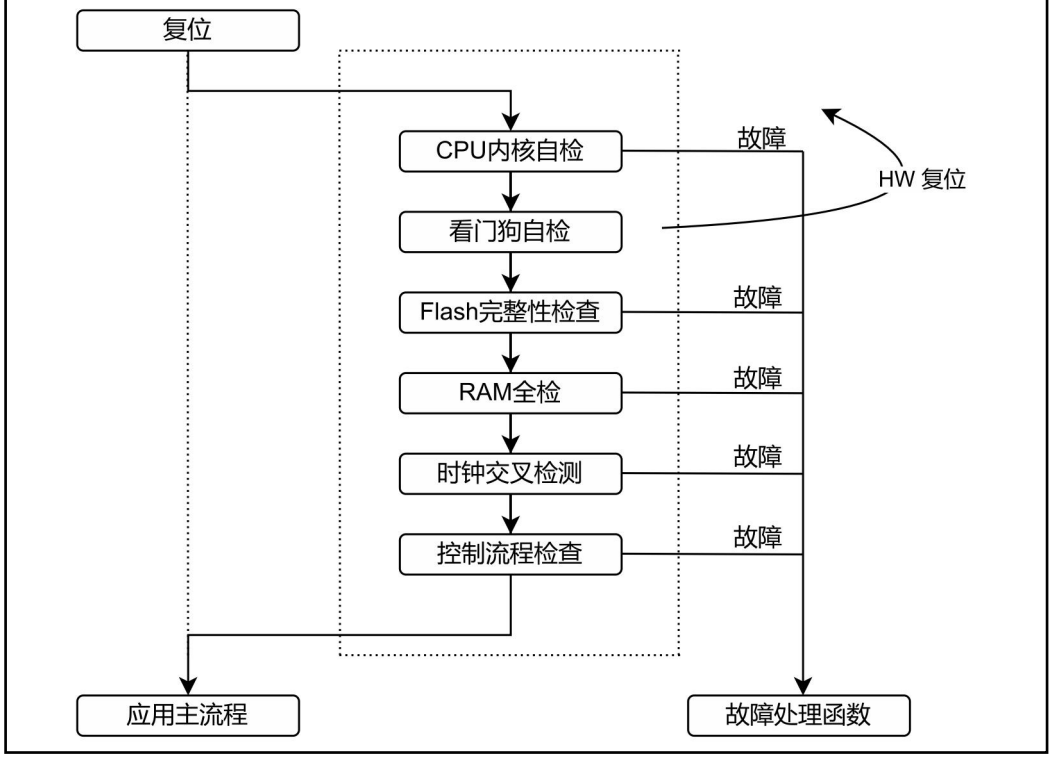

图 4. 启动自检结构

4.1.1 CPU 启动自检

CPU 自检主要是检查内核标志位、寄存器和堆栈指针是否正确。如果发生错误，则将调用故障安全处理函数。这部分检测源代码是用汇编语言编写的，并且在 KEIL 和 IAR 环境之间存在差异。

基本结构如*图 5*所示。

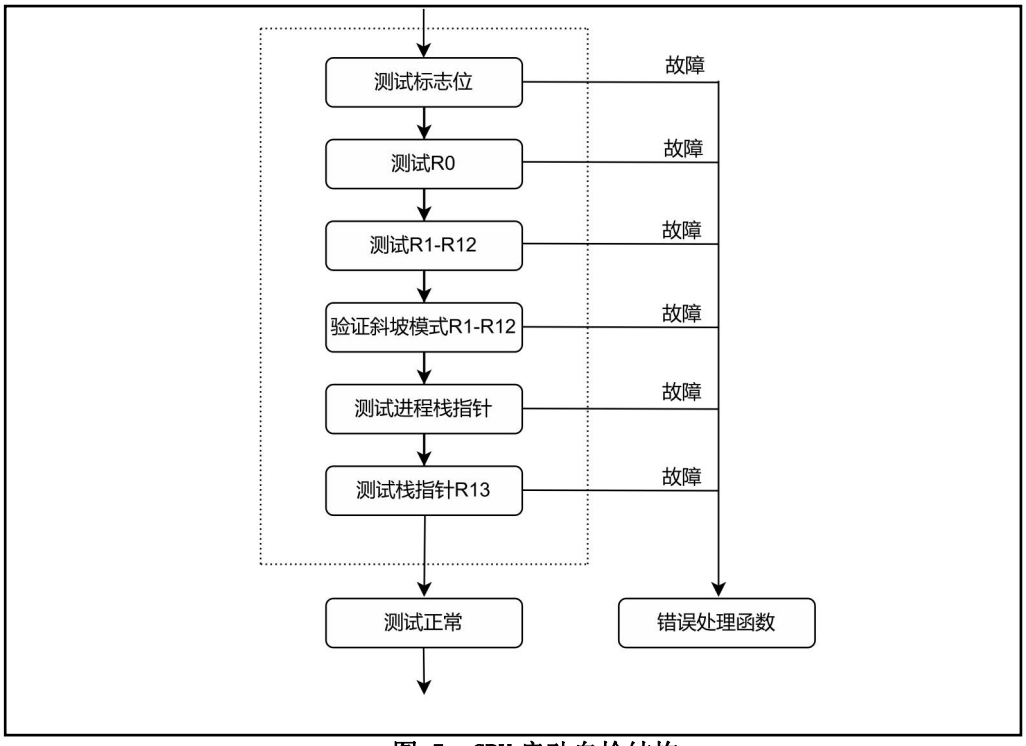

图 5. CPU 启动自检结构

4.1.2 看门狗启动自检

测试结构基于复位状态寄存器内容，通过相应标记寄存所有先前的复位原因，直至寄存器清零（见*图 6*）。

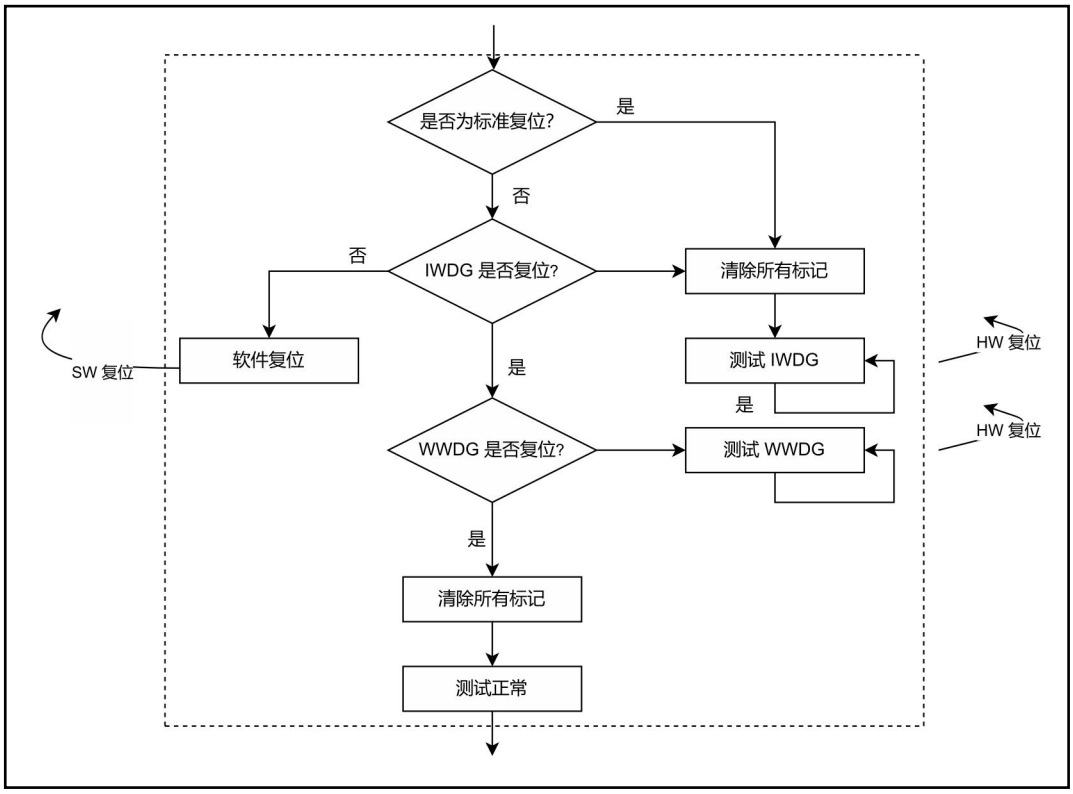

图 6. 看门狗启动自检结构

4.1.3 RAM 启动自检

测试时分 6 个循环，用值 0x00 和 0xFF 逐字交替检查和填充整个 RAM，前 3 个循环按照地址递增执行，后 3 个循环按照地址递减执行。 整个 RAM 检测算法流程如*图 7*所示

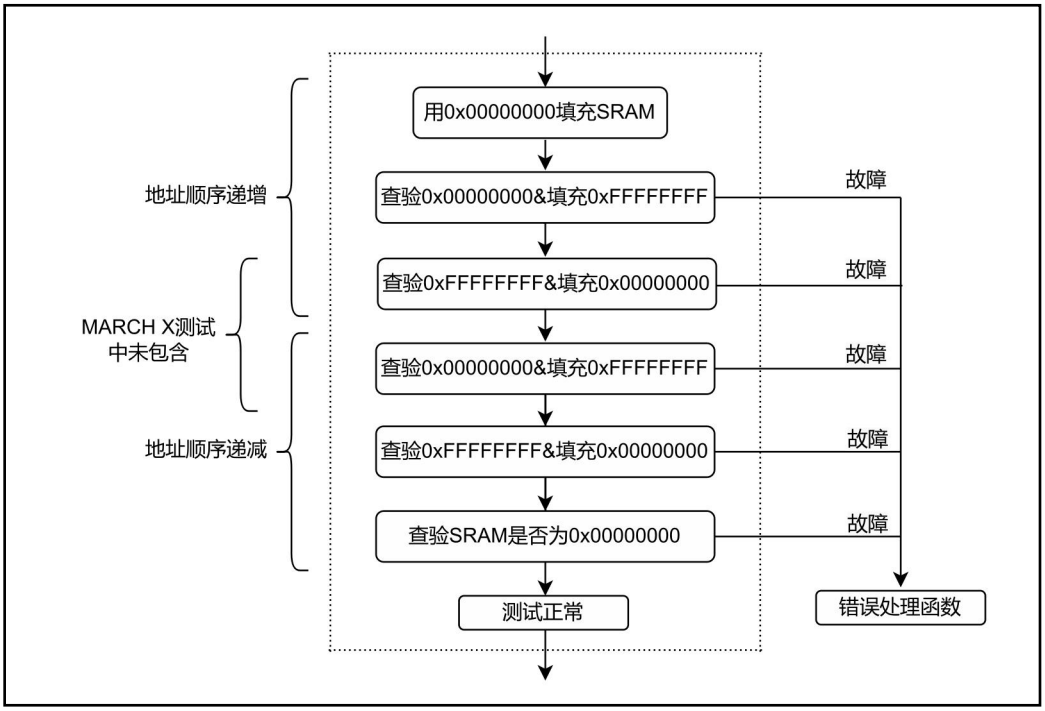

图 7. RAM 启动自检结构

4.1.4 时钟启动自检

测试流程如 *图 8* 所示。

用户指定 HSE 或 HSI 作为系统时钟的时钟源，并可以根据情况选择 HSE、HSI 或 LSI 来验证时钟源是否在正常范围内。当设置 HSI 作为时钟源时，使用 HSI 时钟（-40℃~105℃，+/-2.5%HSI）作为标准捕获 HSE 时钟，并将捕获的 HSE 时钟与标准 HSE 的最大值和最小值进行比较，以确定当前的 HSI 时钟是否在正常范围内；当 HSE 被用作时钟源时，同样适用。

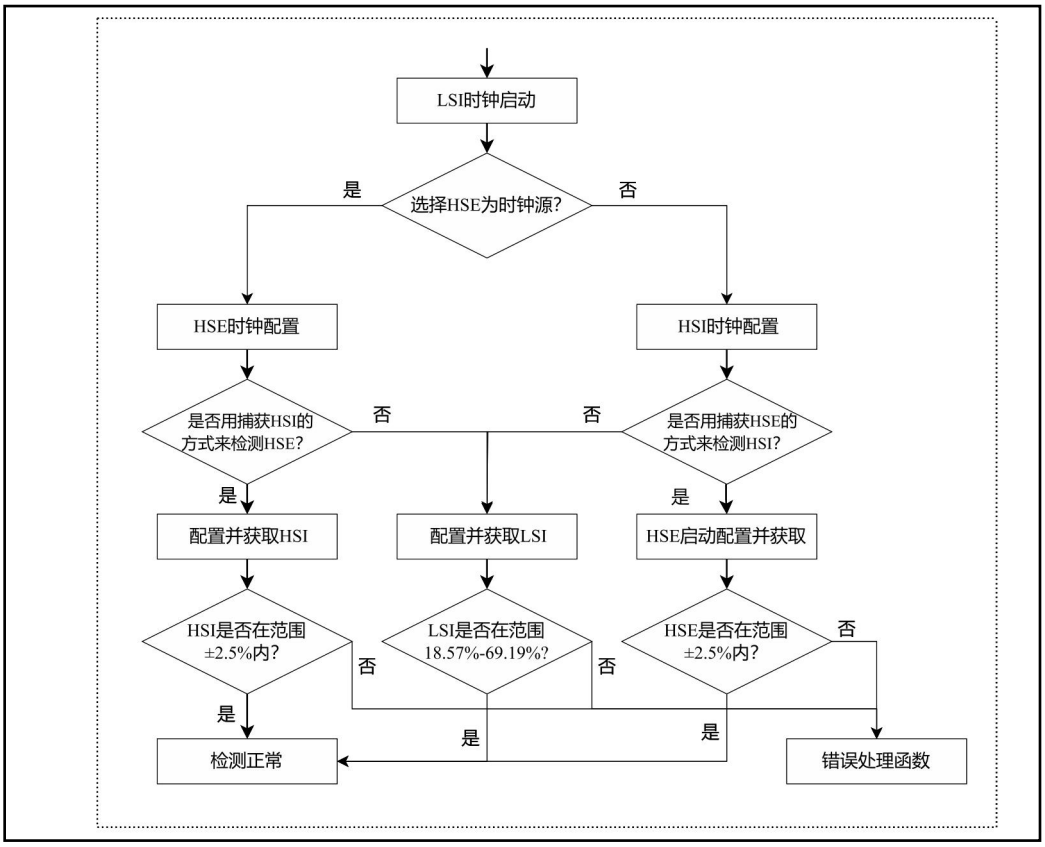

图 8. 时钟启动自检子结构

4.1.5 Flash 存储器完整校验和自检

FLASH 自检是使用 CRC 算法计算 FLASH 中的整个程序和数据存储区，并将结果值与存储在 FLASH 指定位置的预先计算的 CRC 值进行比较的过程（见*图 9*）。

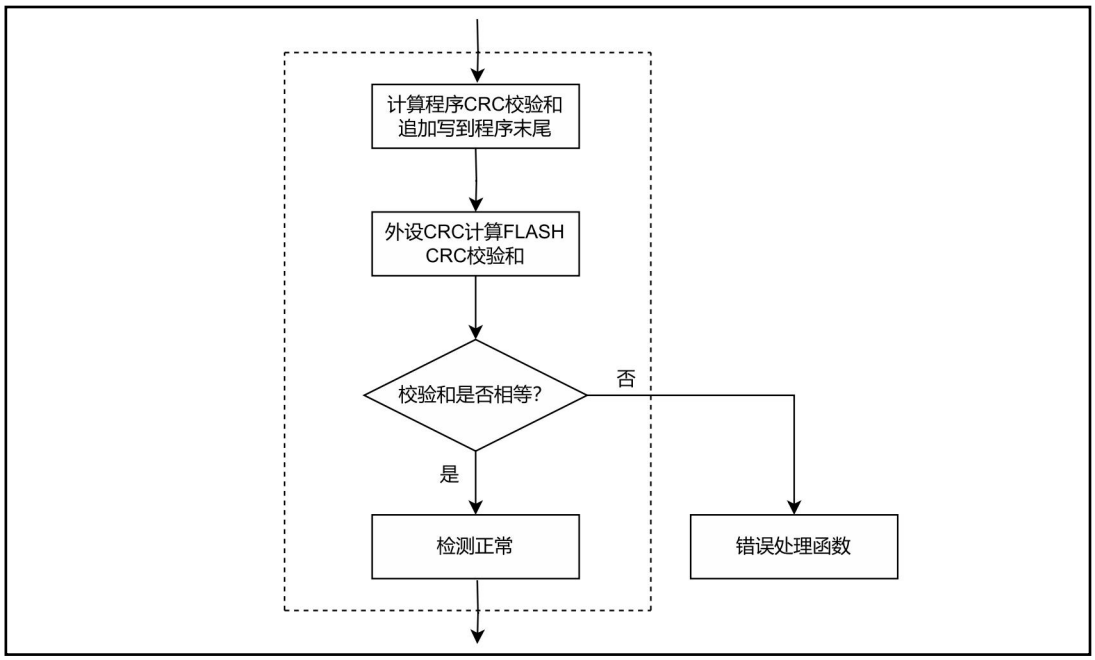

图 9. Flash 启动自检结构

4.2 运行时自检

在运行期间的定期自检必须在进入主循环之前进行初始化。在运行时，使用系统棒作为时间基定期进行检测。

运行时周期检测包括：

- CPU 内核部分运行时测试
- 栈边界上溢测试
- 时钟运行时测试
- Flash 部分 CRC 测试
- 独立和窗口看门狗刷新
- 部分透明 RAM March C/X 测试（在系统中断服务程序中实现）

流程如*图 10*所示:

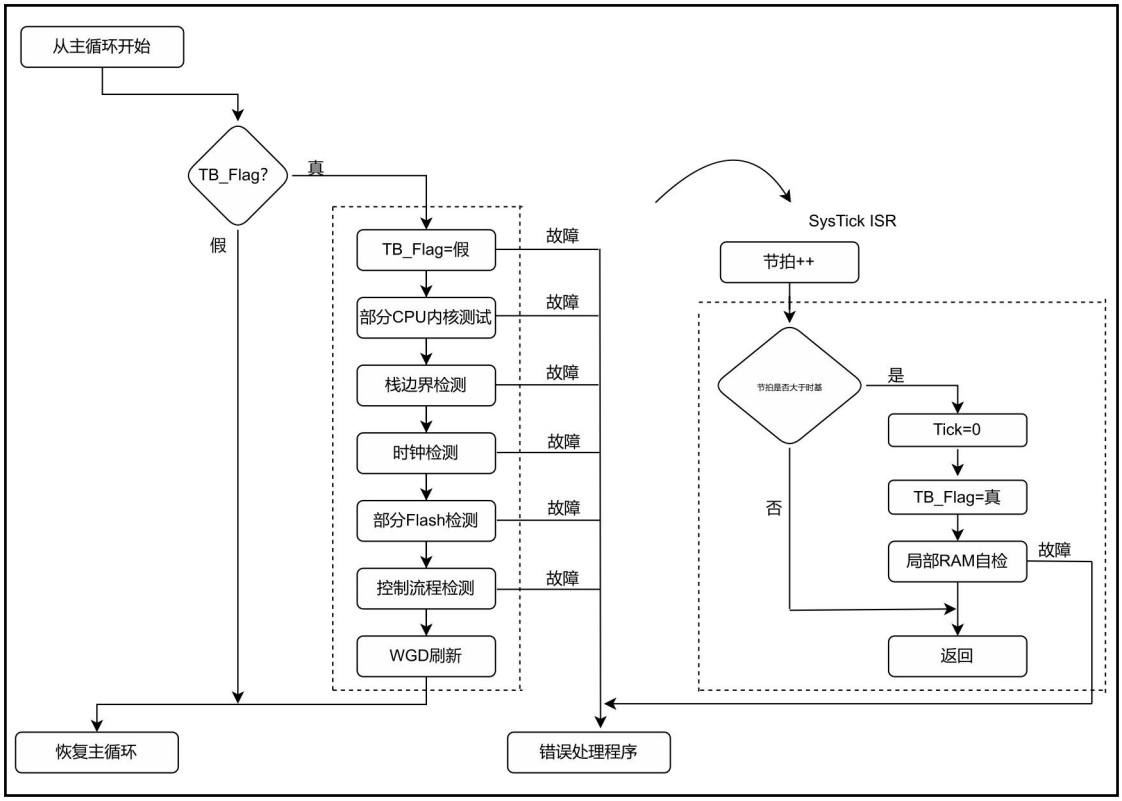

图 10. 周期运行时自检和时基中断服务结构

4.2.1 CPU 运行时自检

CPU 运行时周期自检类似于启动时的自检，运行时不检测内核标志位和堆栈指针（如*图 11*）。

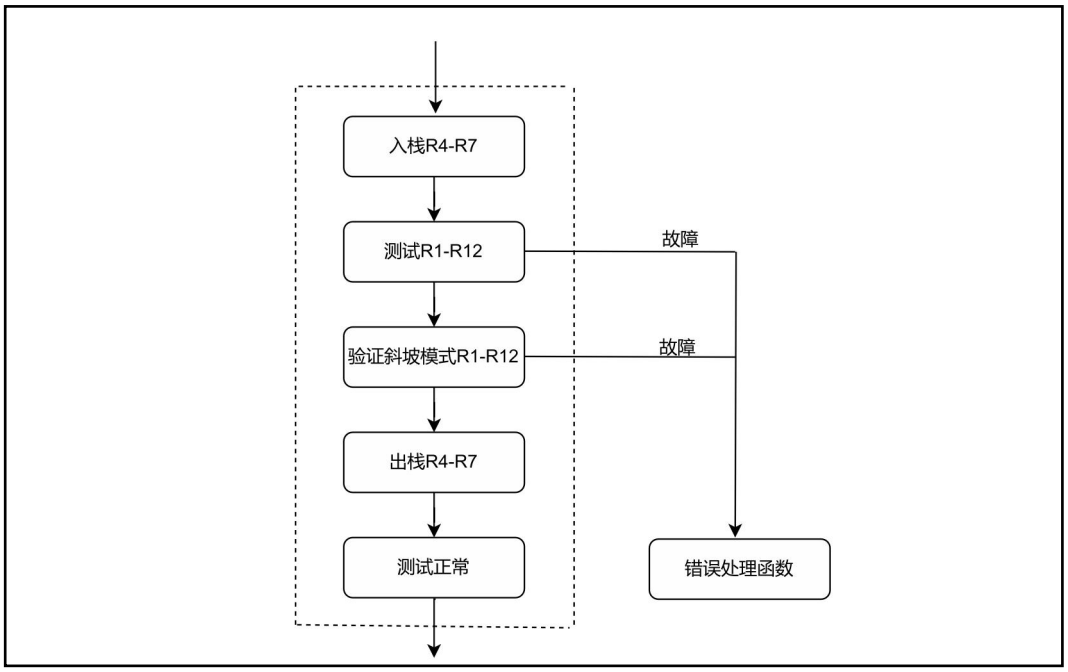

图 11. CPU 运行时自检结构

4.2.2 堆栈边界运行时自检

该测试通过检查特殊数据组合（保存在预留的栈空间顶部）的完整性来检查栈上溢。如果原始数据组合内容损坏，则调用故障保护例程。

数据组置于栈区域预留的最下方地址处。该区域随芯片不同而不同。用户必须定义足够的栈区域并确保正确的数据组合位置（如*图 12*）。

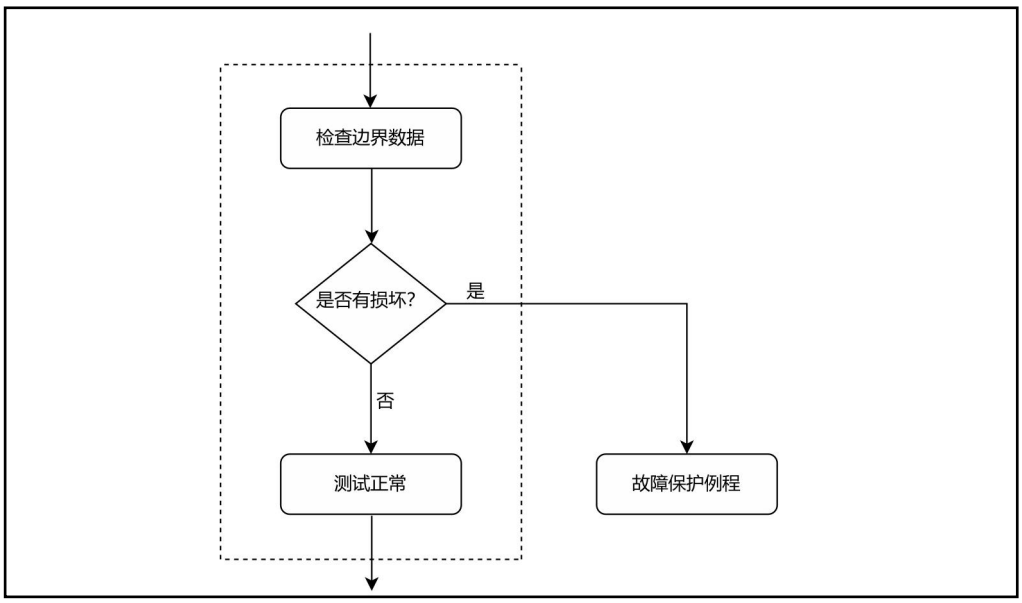

图 12. 栈上溢运行时测试结构

4.2.3 时钟运行时自检

运行时系统时钟检测与启动时时钟检测类似，过程如下（见*图 13*）：

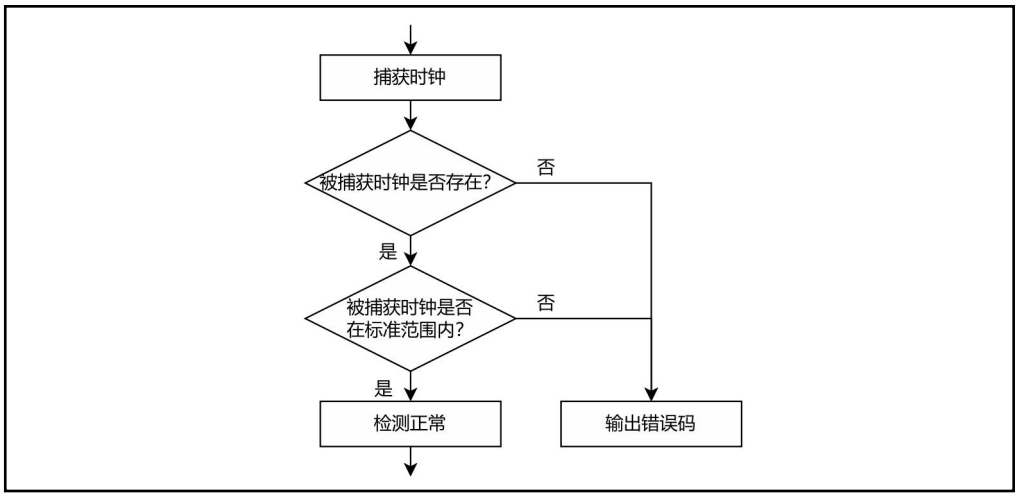

图 13. 时钟运行时自检

4.2.4 局部 FLASH 运行时检测

在运行时执行 Flash CRC 自检。由于不同的检测范围和不同的时间消耗，可以根据用户应用程序的大小配置分段计算 CRC。当计算达到最后一个范围时，将比较 CRC 值。如果不一致，则测试失败（如*图 14*）。

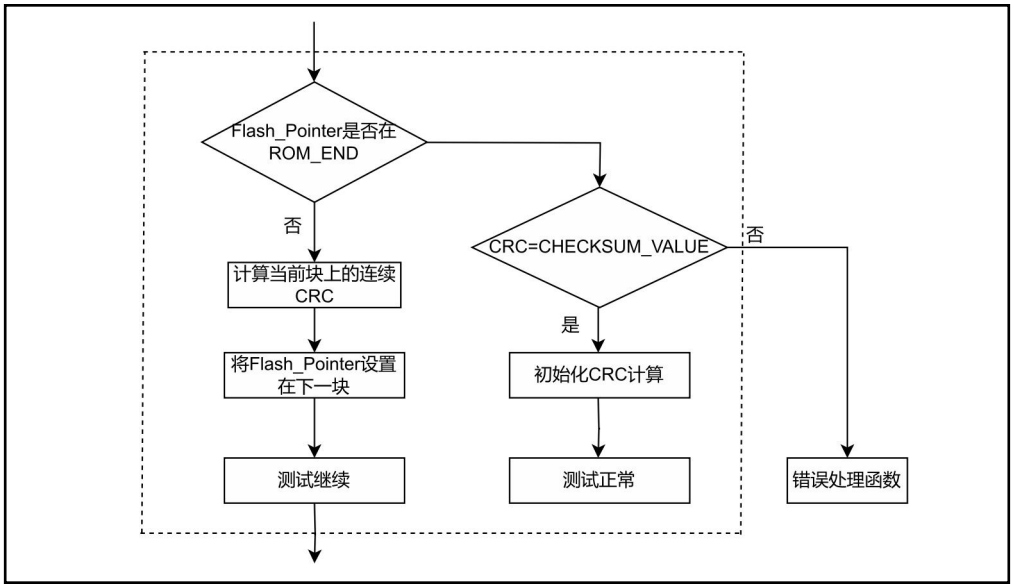

图 14. 局部 Flash 运行时自检结构

4.2.5 部分 RAM 运行时自检

运行时 RAM 自检在系统时基中断函数中执行。该测试只涵盖分配给 B 类变量的内存部分（如*图 15*）。

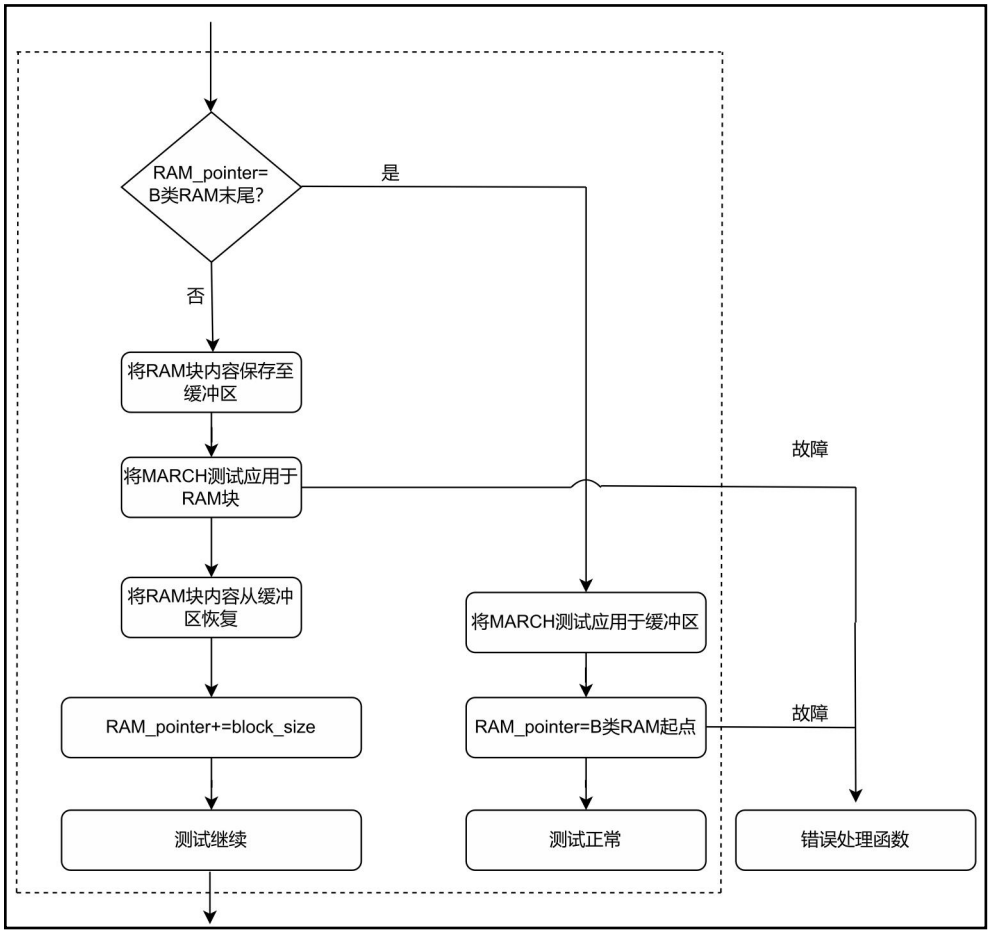

图 15. 部分 RAM 运行时自检结构

5 功能安全生命周期

固件的开发和维护涉及 UL/IEC 60730-1 中关于预防系统错误的要求，主要集中在第 H.11.12.3 节中。所有相关流程都遵循 MindMotion 内部政策，以确保它们具有相应的资质水平。

适用内部规则的以及遵守公认的标准是公认的外部检查机构进行定期检查和审计的目标。

需要有模块级的软件版本管理系统。

5.1 安全要求的规范

内部规划指出了主要目标，即提供一组独立于用户应用程序的通用模块，以便很容易地集成到用户固件中，目标是符合 UL/IEC 60730-1 和 UL/IEC 60335-1 标准。由认证机构审查的所使用的解决方案和方法加快了用户的开发和认证过程。

软件修改是基于一个修改请求，其中详细说明了建议的变更和变更的原因。

5.2 架构规划

STL 数据包结构是长期使用重复认证的 FW 经验的结果，其中模块被集成到过去专门用于不同产品的 MindMotion 标准外围设备库中。新 FW 的主要目标是消除对不同产品的任何依赖，并将安全依赖部件与基于为整个 MindMotion 家族开发的新的独特硬件抽象接口（HAL）集成到一个通用的自检栈中。

从系统的角度来看，这种通用体系结构要安全得多，在现有或迁移到新产品时，更容易维护和集成解决方案。许多客户在许多不同的配置中应用了相同的结构，因此他们的反馈是非常重要的，如果有的话，还有助于有效地解决弱点。

技术规范的修改需要经过开发团队会议确认。

5.3 模块规划

模块测试方法来源于原始 FW 所用的已证明解决方案。一些方法经过优化，用于加速测试周期，从而应用这些自检方法来最小化最终应用中进程安全时间的限制，这些自检方法大多由软件提供。

5.4 编码

编码基于 MindMotion 内部策略定义的原理，遵循公认的国际编码标准、验证工具和编译器。强调执行非常简单、单一和透明的线性编码结构，在使用简化而清晰的输入和输出时，逐步调用定义的测试功能组。流程由特定的流程控制机构提供保护，并与系统节拍中断服务同步，从而提供特定的特殊后台检测。一旦所有的局部检查程序成功完成，应专门提供硬件看门狗服务。

5.5 测试模块

模块已经使用不同的开发工具对不同的产品进行了功能测试。
修改要求后的验证需要适当的和确定的测试用例。

5.6 模块集成测试

模块集成已经在几个专门针对使用不同开发工具的不同产品的例子中进行了测试，重点是适当的时间测量、代码控制流、堆栈使用和其他方法。

修改的评估是根据指定的验证和验证活动进行的，其中可能包括：

- 对更改后的软件模块的重新验证
- 对受影响的软件模块的重新验证
- 对完整系统的重新验证

5.7 维护

对于 FW 维护 MindMotion 使用来自根据标准内部流程处理的客户（包括初步测试人员）的反馈。新的升级会定期发布，或在发现一些重大错误时发布。所有版本都与描述解决方案及其集成方面的适当文档发布。升级、应用的修改和已知限制之间的差异在包中包含的相关版本说明中描述。特定的工具用于支持适当的软件修订编号、源文件和相关的文档归档。

6 软件库移植

6.1 软件库目录结构

(如*图 16*所示)

KEIL IAR

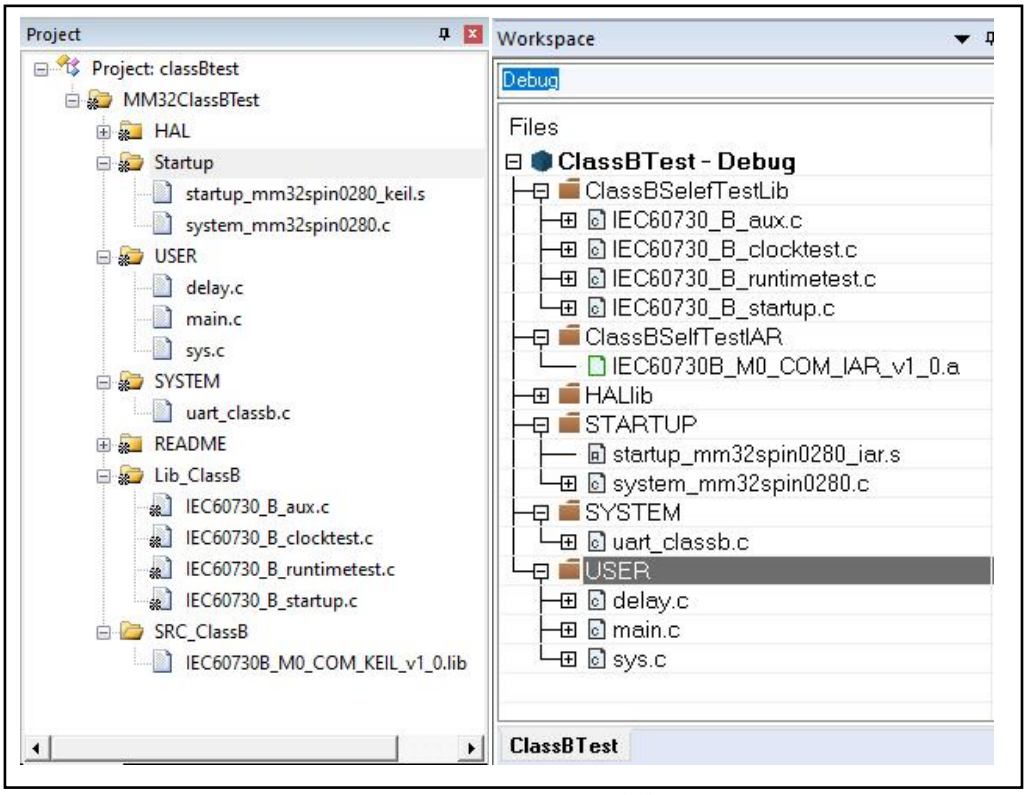

图 16. CLASS B 目录结构

6.2 B 类功能安全软件库的移植

演示基于 MM32SPIN0280 平台

1. 将 ClassBTest 文件夹拷贝到实际应用工程里，ClassBTest 目录下包含 Library 目录，Library 目录包含移植所需的库文件、C 源文件和头文件（如*图 17*）；

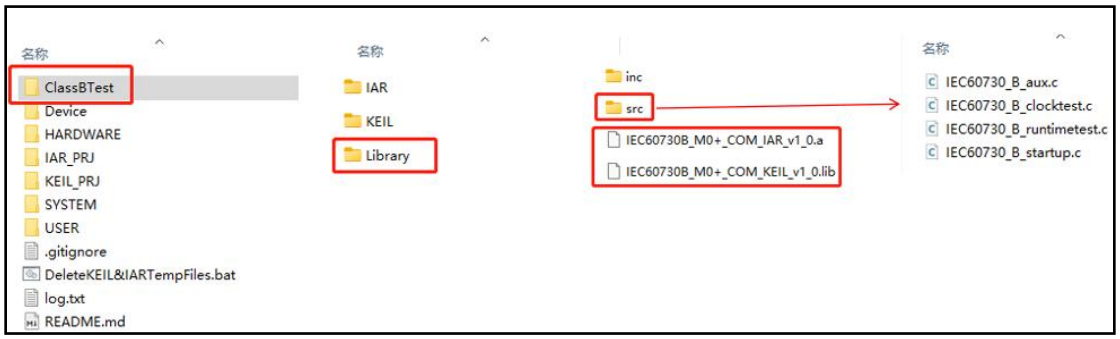

图 17. Library 目录

2. 将 Library 目录包含的移植所需的库文件、C 源文件添加进 MDK 或 IAR 的工作组中,在实际工程中添加头文件路径（如*图 18*）；

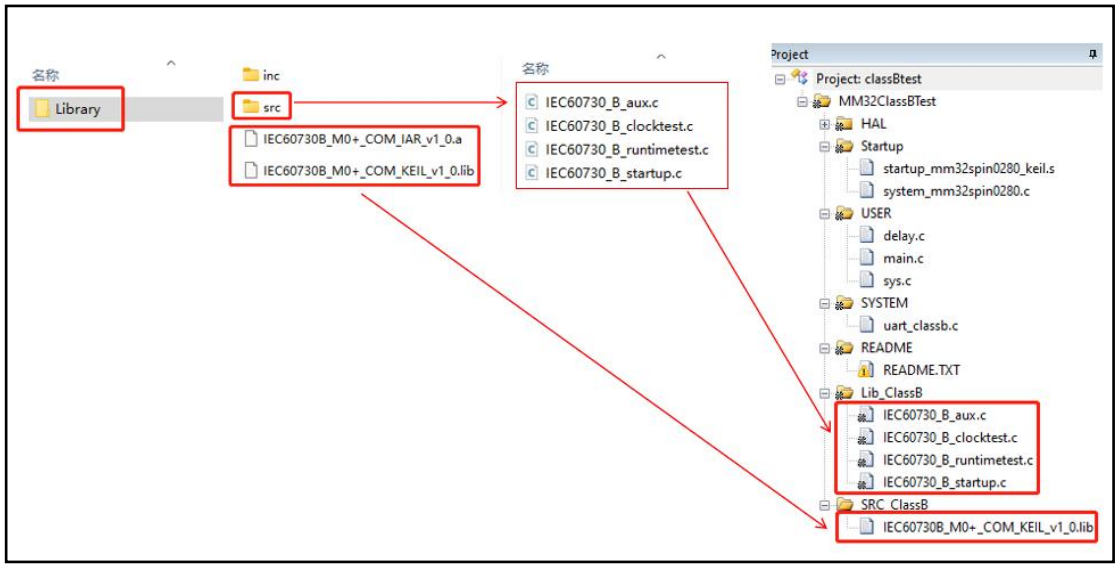

图 18. 所需添加文件

3. 启动时的检测代码在 MDK 中位于 IEC60730_B_startup.c 文件的$Sub$$main 函数中，IAR 中要修改启动文件中的项目入口地址到 IEC60730_B_startup 函数（如*图 19*）；

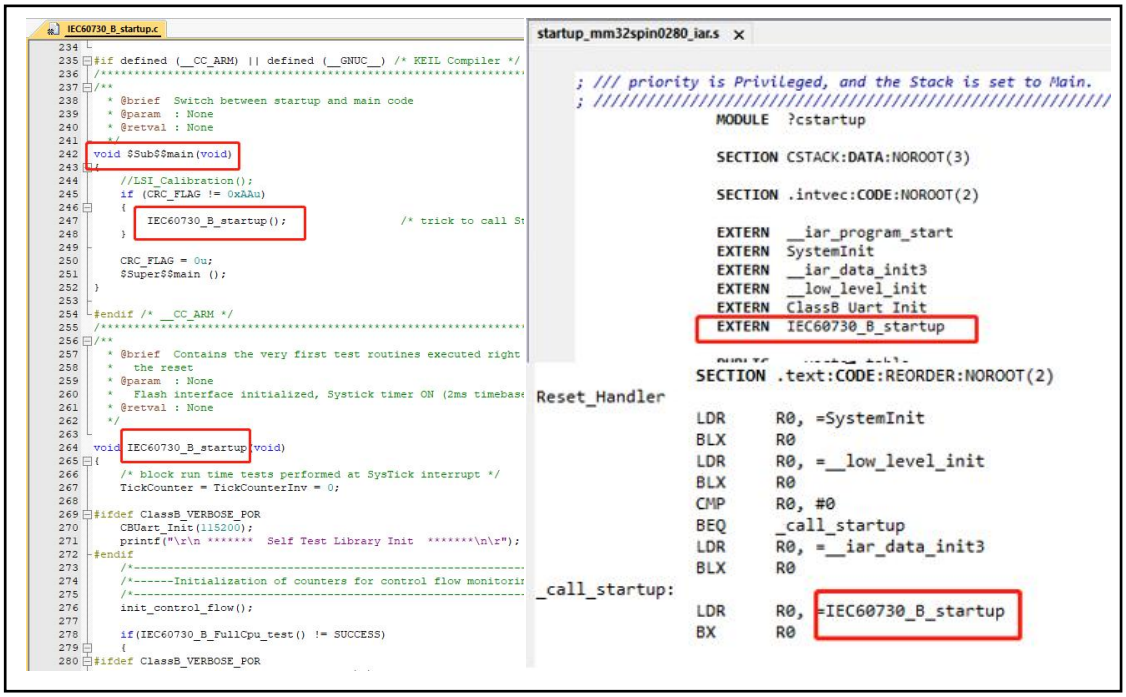

图 19. 启动检测的入口

4. 运行时检测在 main.c 中 while(1)循环前调用 IEC60730_InitRunTimeChecks()函数，在 while(1)循环中调用 IEC60730_DoRunTimeChecks()函数（如*图 20*）；

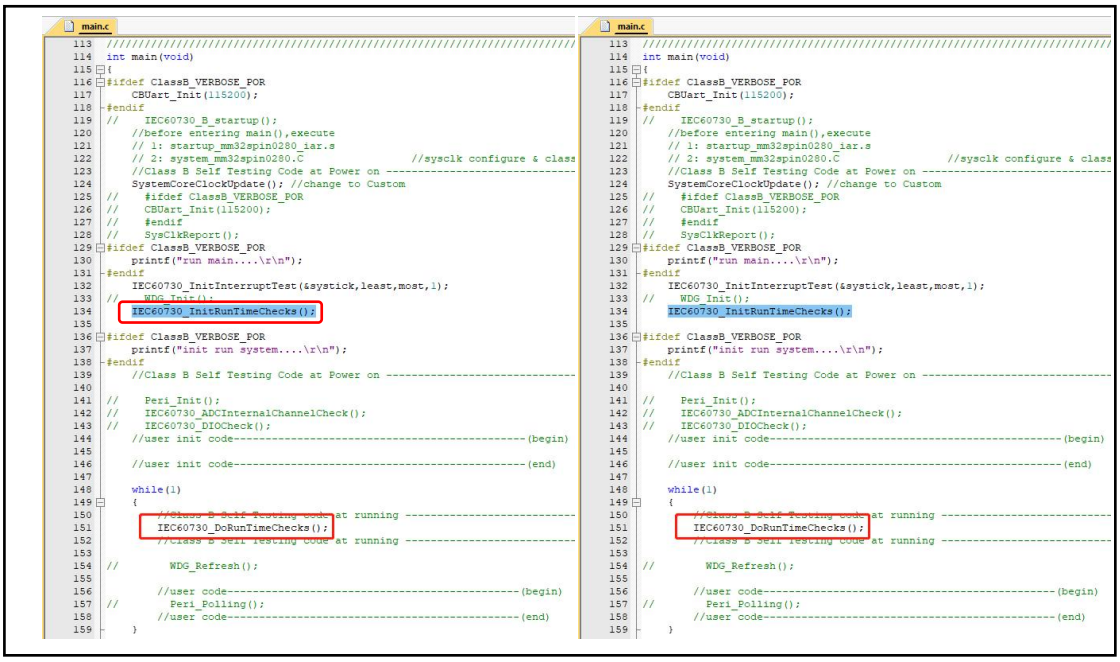

图 20. 运行时检测

5. 将 KEIL_PRJ 文件夹中的 ClassBtest.sct， crc_gen_keil.bat， crc_load.ini，MM32_CRC.exe 四个文件拷贝到实际应用的.uvprojx 所在目录；在 user 配置中设置 crc_gen_keil.bat 运行方式（如*图 21*）；

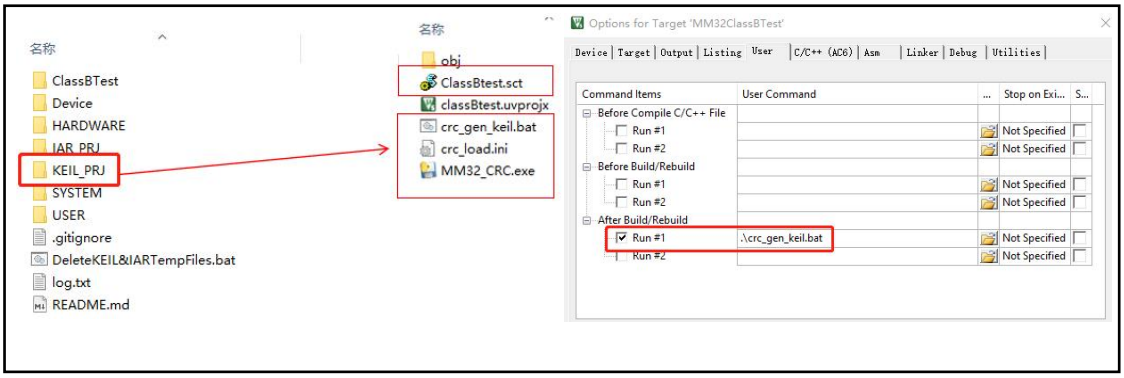

图 21. keil 平台 FLASH 检测所需文件

6.Utilities 选项中设置 crc_load.ini 路径用于 debug 时加载含 CRC 的 hex 文件，linker 选项中取消勾选默认分散加载文件，设置例程中的 SCT 文件路径（如*图 22*和*图 23*）；

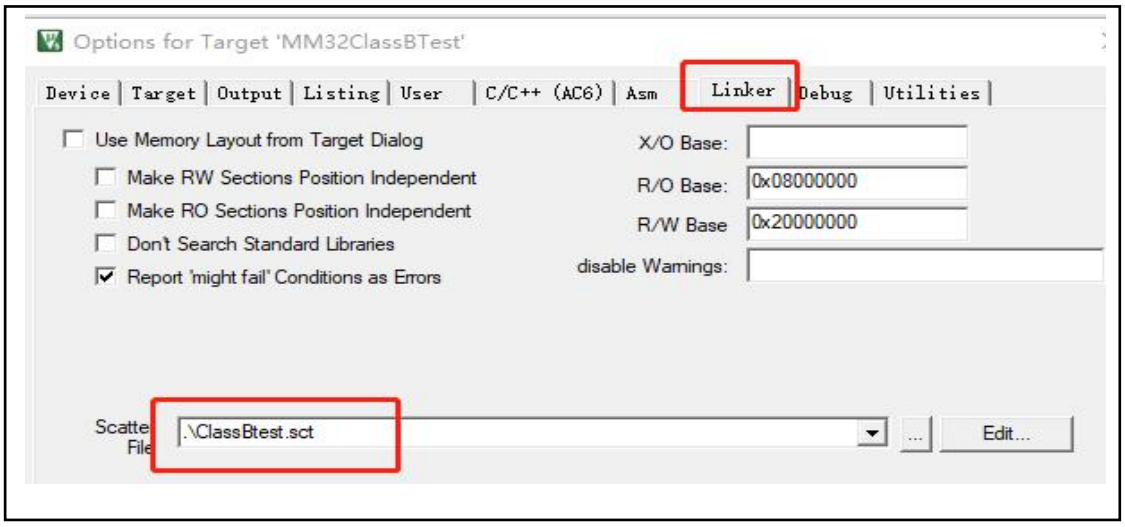

图 22. keil 平台 SCT 文件路径配置

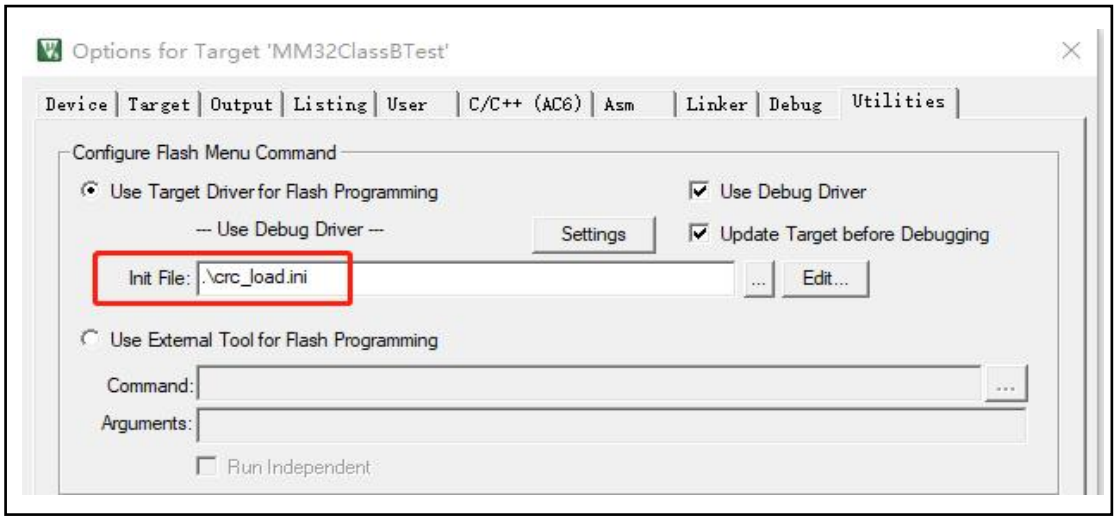

图 23. keil 平台调试脚本配置

7. IAR 下将 IAR_PRJ 文件夹中的 MM32SPIN0280_CLASSB.icf，crc_gen_keil.bat，MM32_CRC.exe 三个文件拷贝到实际应用的.eww 所在目录（如*图 24*）；

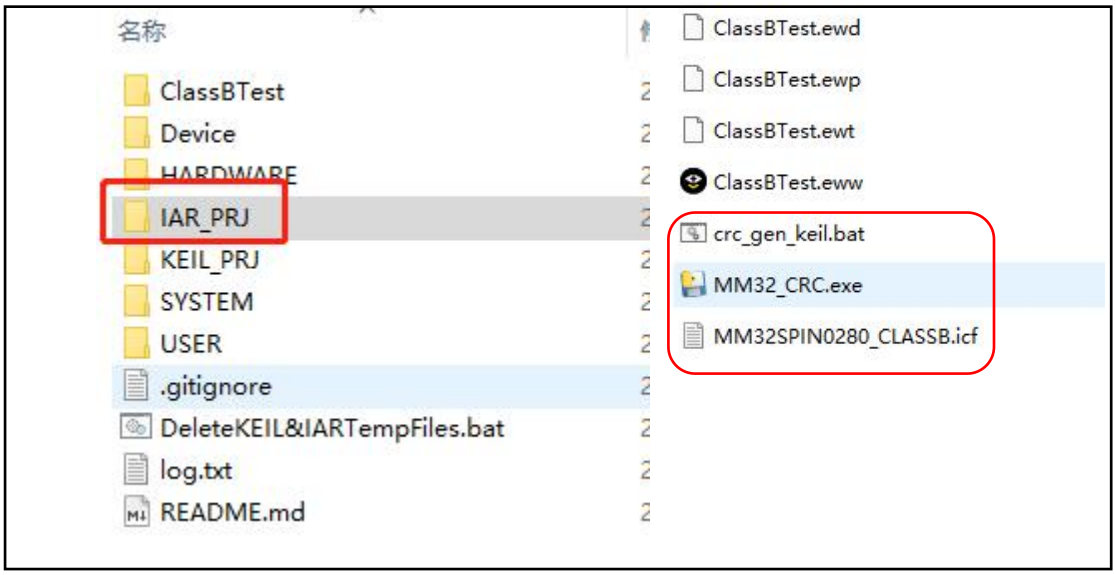

图 24. IAR 平台 FLASH 检测所需文件

8. 在 Build Action 配置中设置 crc_gen_keil.bat 运行方式，linker 选项中设置例程中的 icf 文件路径（如*图 25* 和*图 26*）；

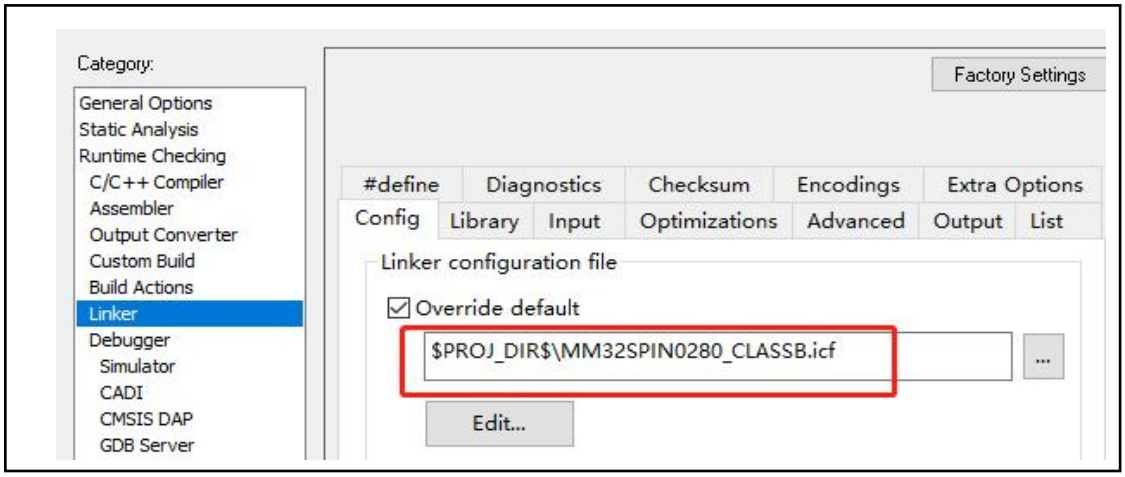

图 25. IAR 平台 ICF 文件路径配置

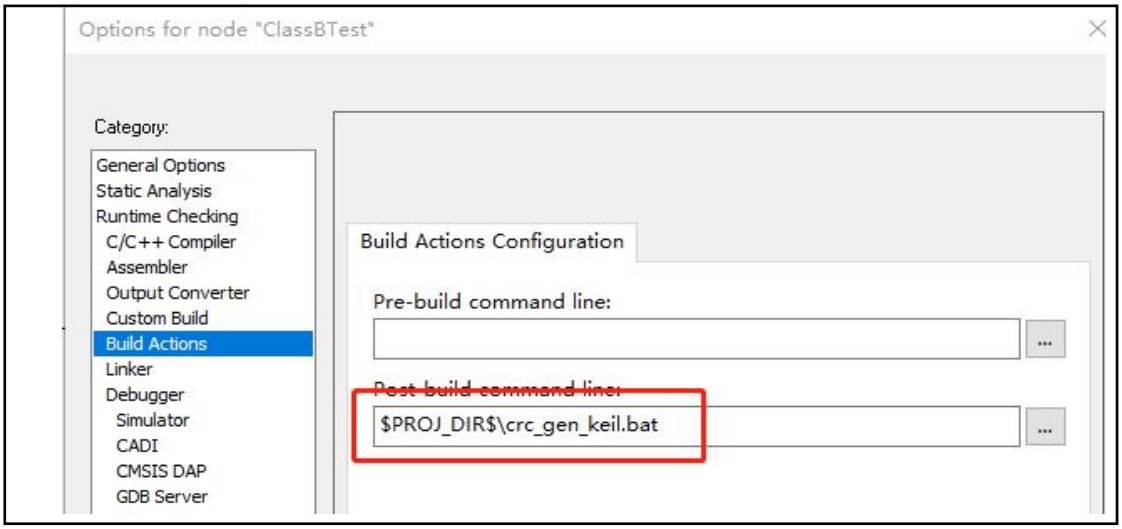

图 26. IAR 平台 CRC 脚本文件路径配置

9. crc_gen_keil.bat 文件中可以修改实际应用的工程名称以及工程输出文件的路径；若修改了工程名称，crc_load.ini 文件中的可以执行文件的名称要一并修改（如*图 27*）；

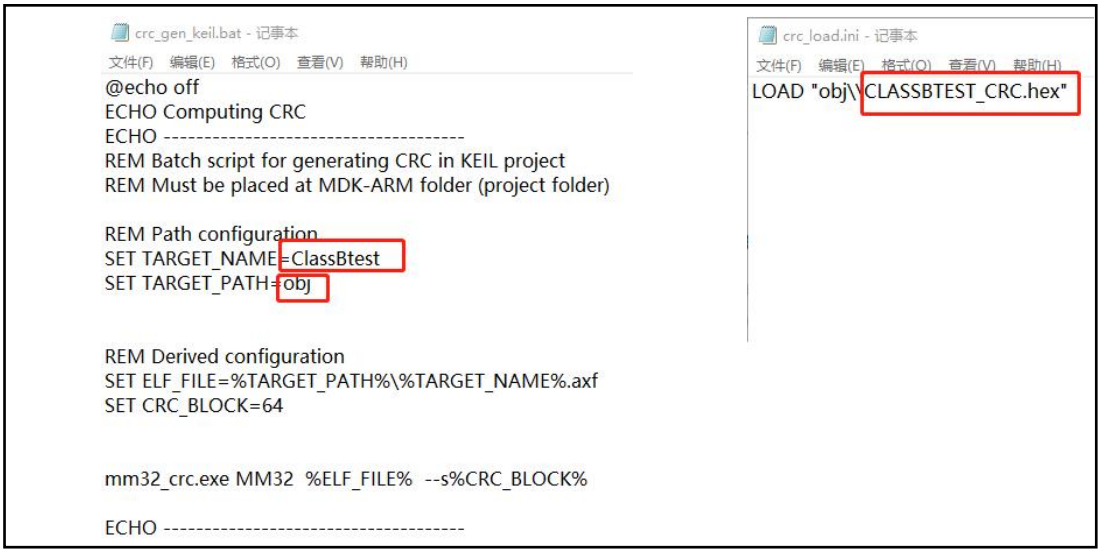

图 27. keil&IAR 平台 CRC 脚本文件中工程名配置

10. 在 IEC60730_B_startup.c 文件中设置相应型号芯片的片上 RAM 大小，以及运行时 RAM 的检测区域，class b 变量存放地址，在对应的 SCT 或 icf 文件中做相应修改（如*图 28*、*图 29* 和 *图 30*）；

IEC60730_B_startup.c

```
79  extern uint32_t Load$$LR$$LR_IROM1$$Base,Load$$LR$$LR_IROM1$$Length,Load$$LR$$LR_IR
80  #define CRC_BLOCK_BYTE    64u
81  #define ROM_START         ((uint32_t)&Load$$LR$$LR_IROM1$$Base)
82  #define CODE_SIZE         ((uint32_t)&Load$$LR$$LR_IROM1$$Length)
83  #define CODE_END          ((uint32_t)&Load$$LR$$LR_IROM1$$Limit)
84  #define ROM_END           ((uint32_t)((CODE_END & (CRC_BLOCK_BYTE - 1))?((CODE_END
85  #define ROM_SIZE          (ROM_END - ROM_START)
86  #define ROM_SIZEinWORDS (uint32_t) (ROM_SIZE/(CRC_BLOCK_BYTE/16))
87  #define STEPS_NUMBER          ((uint32_t)ROM_SIZE/CRC_BLOCK_BYTE)
88  #define FLASH_BLOCK_WORDS     (uint32_t)((ROM_SIZEinWORDS)/STEPS_NUMBER)
89  #define REF_CRC32  ROM_END
90  #endif  /* __CC_ARM */
91
92  #define MCU_RAM_BuffertestEnd_ADDR    (0x20000000UL + 32UL-1UL)
93  #define MCU_RAM_RunTimetestStart_ADDR (0x20000030UL)
94  #define MCU_RAM_RunTimetestEnd_ADDR   (0x2000007FUL)
95  #define MCU_RAM_End_ADDR    (0x20001FFFUL)
96  uint32_t RAMRunTimetest_start;
97  uint32_t RAMRunTimetest_end;
98  uint32_t RAMBuffertest_end;
99  uint32_t RAM_endadd_run;
100
```

图 28. RAM 检测区域配置

```
18
19      RAM_BUF 0x20000000 UNINIT 0x20 {; Run-time transparent RAM test
20        IEC60730_B_startup.o (RUN_TIME_RAM_BUF)
21      }
22
23      RAM_PNT 0x20000020 UNINIT 0x10 {; Run-time transparent RAM test
24        IEC60730_B_startup.o (RUN_TIME_RAM_PNT)
25      }
26
27      CLASSB 0x20000030 UNINIT 0x28 {         ; Class B variables
28      IEC60730_B_startup.o (CLASS_B_RAM)
29      }
30
31      CLASSB_INV 0x20000058 UNINIT 0x28 {              ; Class B inve
32      IEC60730_B_startup.o (CLASS_B_RAM_REV)
33      }
```

图 29. SCT 文件中 RAM 检测区域修改

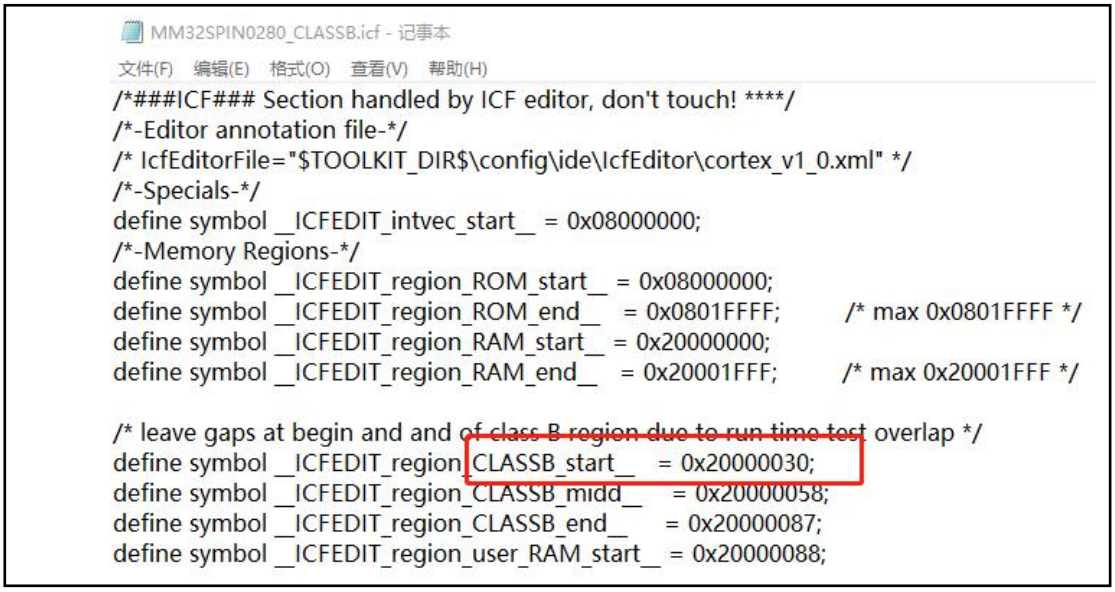

图 30. ICF 文件中 RAM 检测区域修改

11. 启动以及运行时的时钟检测部分可以根据实际应用的需求修改例程中所用到的硬件 timer 资源，例程中 Timer2 捕获内部 LSI 或者外部 HSE 的 128 分频输入；Timer3 捕获 A8 引脚的 MCO 输出，MCO 输出配置为内部 HSI 输出（如图 31、图 32、图 33 及图 34）；

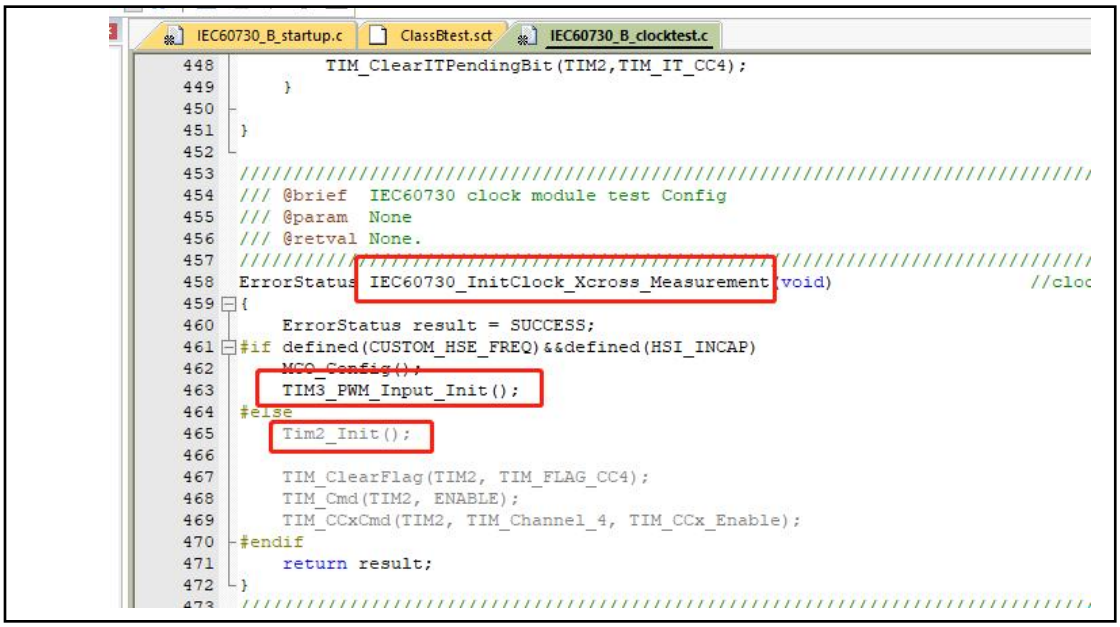

图 31. 时钟检测部分代码

16.5.19 TIM2_OR 输入选项寄存器

偏移地址：0x50

复位值：0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.								TI4_RMP		Res.				ETR_RMP	
								rw	rw					rw	rw

Bit	Field	Description
15:8	Reserved	保留，必须保持复位值。

图 32. 内部 Timer 输入选项寄存器

Bit	Field	Description
7:6	TI4_RMP	TI4 复用 00：CH4 GPIO 或比较器输入 01：LSI 时钟输入 10：保留 11：HSE_CLK_DIV_128 时钟输入

图 33. 内部 Timer 输入选项选择

5.2.4.10 微控制器时钟输出（MCO）

微控制器时钟输出（MCO）允许时钟输出到外部 MCO 引脚上。相应 GPIO 端口的配置寄存器必须被配置为复用输出功能。可以选择以下五个时钟信号中的一个作为 MCO 输出时钟：

表 5-2 MCO 与时钟源对应关系

时钟配置寄存器（RCC_CFGR）的 MCO 位	时钟源
010	LSI
100	SYSCLK
101	HSI
110	HSE
111	PLL/2

图 34. 内部 MCO 输出选项选择

12.运行时的 CPU、FLASH、时钟、堆栈的检测周期可以在 IEC60730_B_param.h 文件中修改，时钟的捕获选择也可以在 IEC60730_B_param.h 中设置；运行时的 RAM 检测在 IEC60730_B_aux.c

文件中的 1ms 时基中断中处理，因 RAM 运行时检测会关闭中断，可根据需要将 RAM 运行时检测功能放置到更高优先级的中断中（如*图 35* 和*图 36*）；

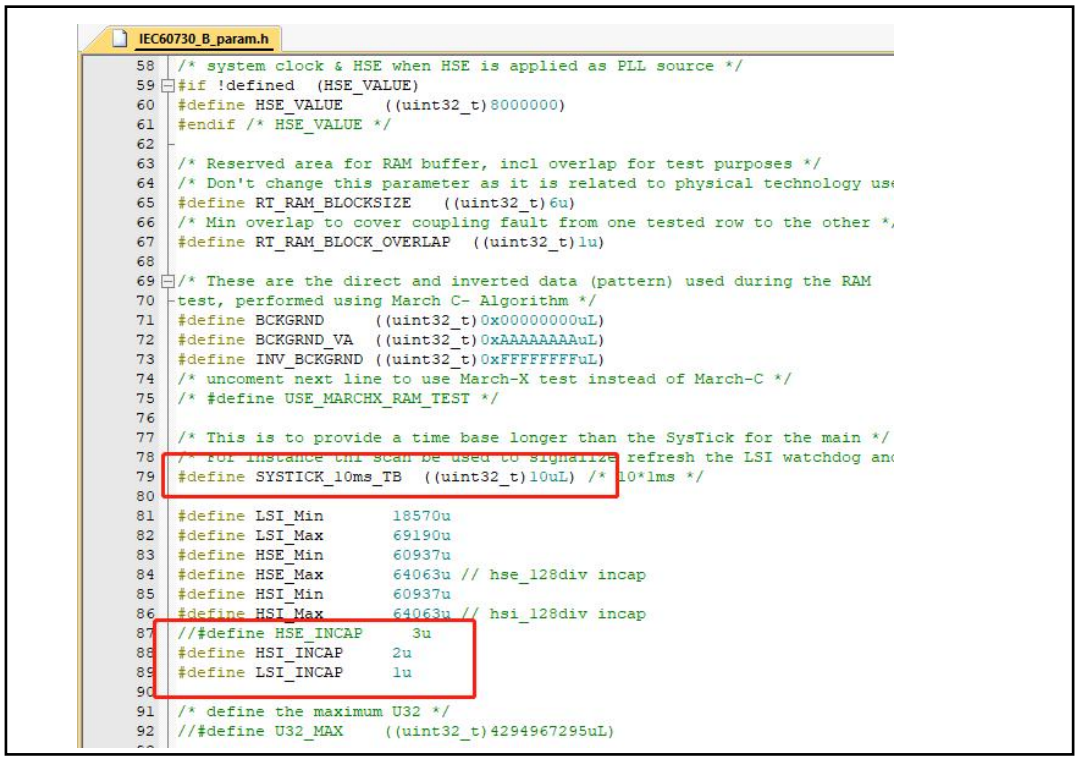

图 35. Class B 参数配置

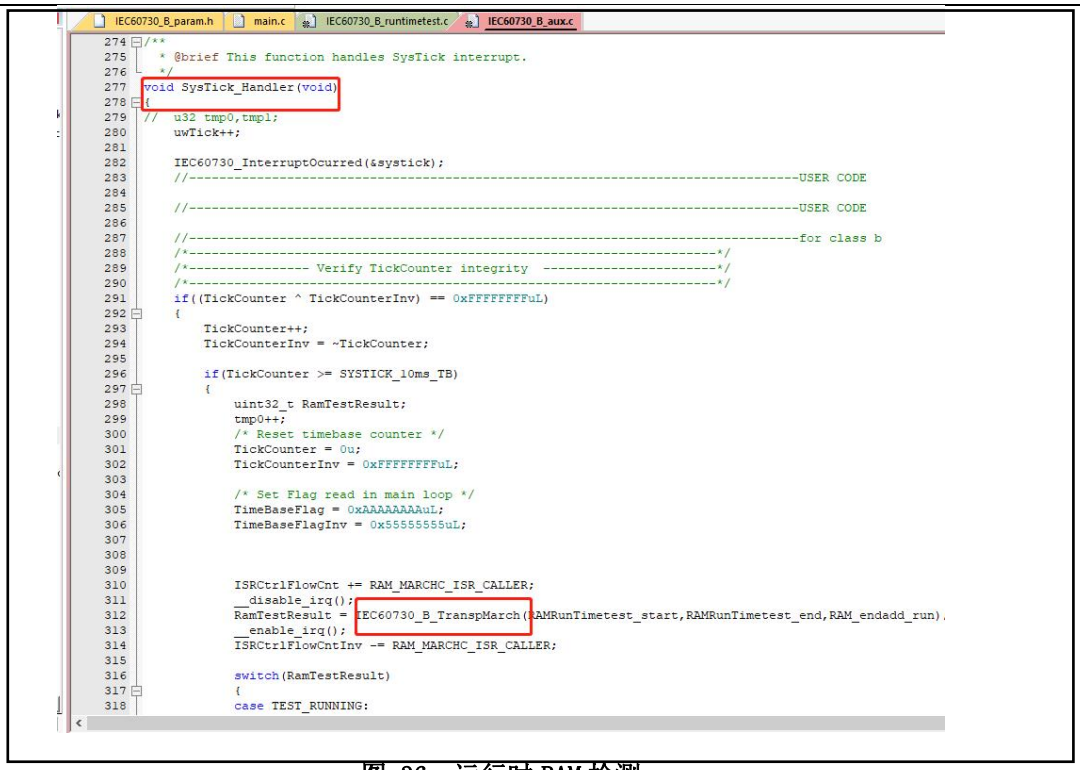

图 36. 运行时 RAM 检测

13.例程的 FLASH 检测采用 CRC 校验的方式，默认使用 CRC 外设；对于片上无 CRC 外设的芯片，可以修改 IEC60730_B_startup.c 以及 IEC60730_B_runtimetest.c 中 FLASH 检测的函数接口，改为软件方式的 CRC 校验（如*图 37* 和*图 38*）；

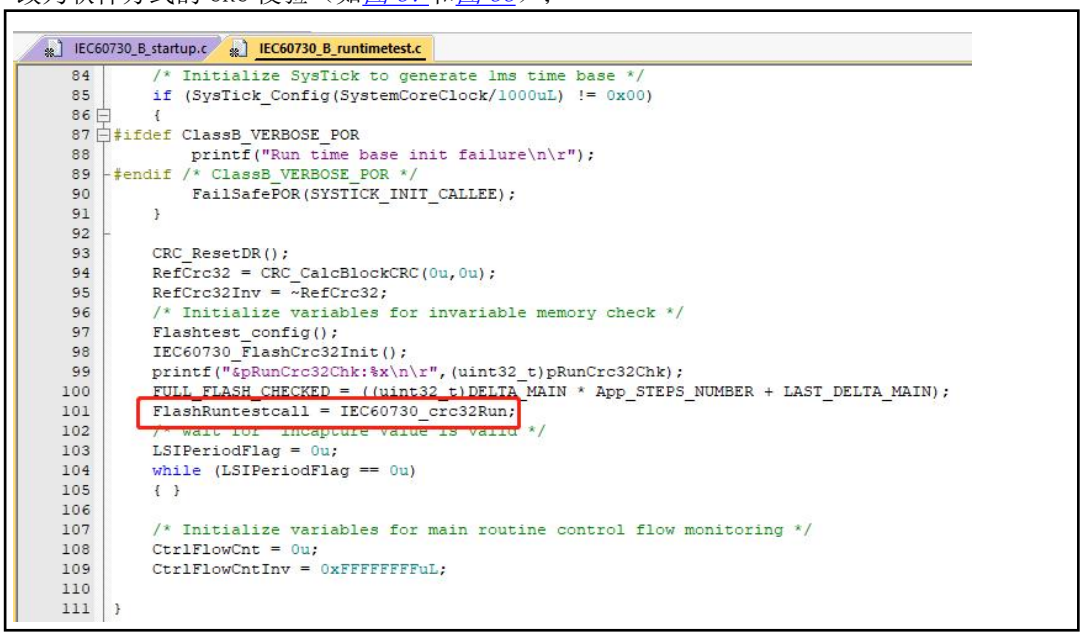

图 37. 启动时 FLASH 检测的函数接口

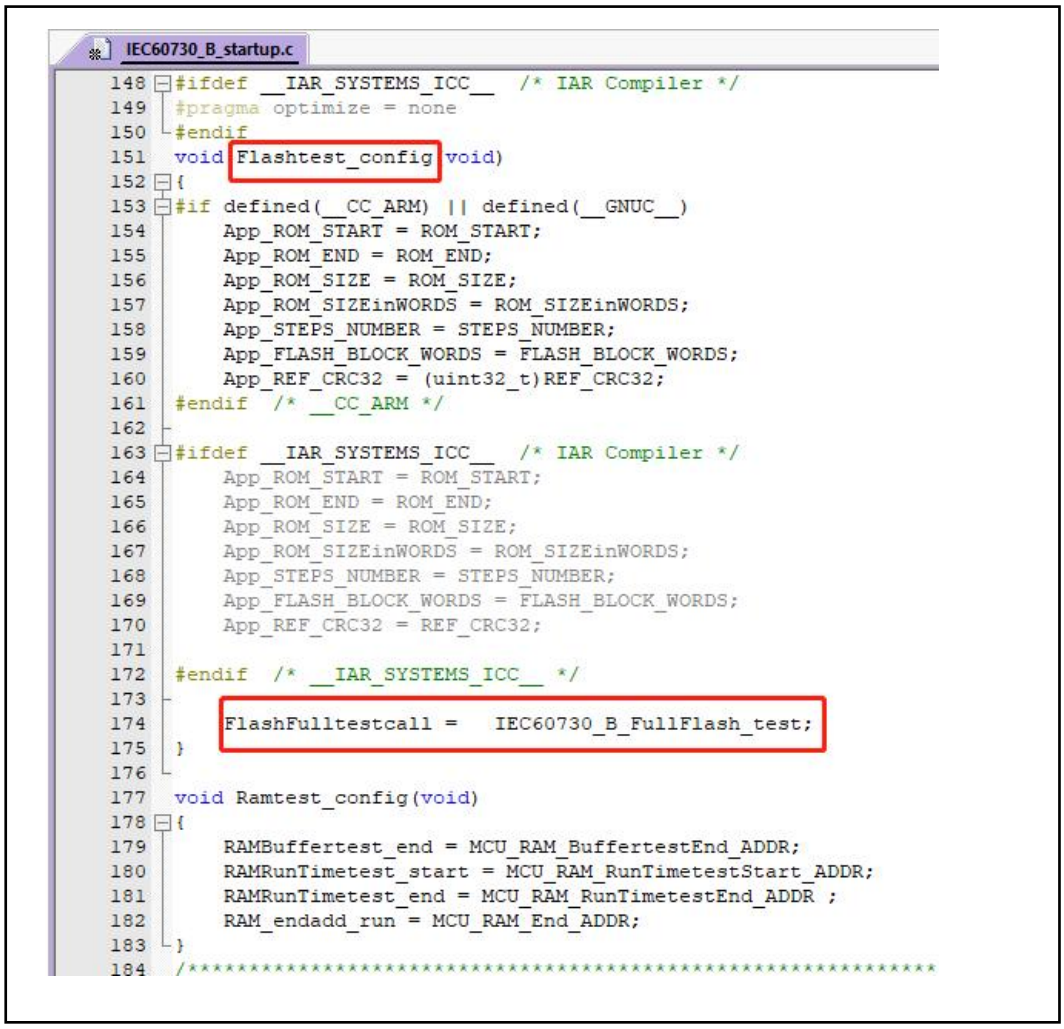

图 38. 运行时 FLASH 检测的函数接口

14. 例程正常运行时打印信息如*图 39*和*图 40*所示；

```
 *******  Self Test Library Init  *******
 Start-up CPU Test OK
Pin reset
SW reset
...  Power-on or software reset,  testing IWDG ...

 *******  Self Test Library Init  *******
 Start-up CPU Test OK
Pin reset
IWDG reset
...  IWDG reset from test or application,  testing WWDG

 *******  Self Test Library Init  *******
 Start-up CPU Test OK
Pin reset
IWDG reset
WWDG reset
...  WWDG reset,  WDG test completed ...
ROM size = 10a0    0x08000000   0x08004280   0x00004280
FLASH 32-bit CRC Address and Value  :8004280   bcc85794
 Start-up FLASH 32-bit CRC OK
 Control Flow Checkpoint 1 OK
 Full RAM Test OK
STL_ClockStartUpTest

Read Incapture Frequency OK freq = 40851Hz !
 Clock frequency OK
 Control Flow Checkpoint 2 OK
run main....
init run system....
```

图 39. Debug 版本打印信息

```
[18:01:17.721]收←◆run main....
&pRunCrc32Chk:8000000
init run system....
```

图 40. Release 版本打印信息

7 历史版本

日期	版本	变化说明
2023 年 10 月 31 日	V0.1	初始内试