STM32U5 系列 IEC 60730 自测库用户指南

引言

本文档适用于包含 Arm® Cortex®-M33 内核的 STM32U5 系列微控制器的 X-CUBE-CLASSB 自检库集合。订购代码 X-CUBE-CLASSB。

安全在电子应用中至关重要。随着组件安全要求级别的稳步上升，电子设备制造商在其设计中包含了许多新的技术解决方案。用于提升安全性的技术在持续发展，并时常被纳入安全标准的更新版本中。

各种权威机构发布的全球标准中规定了当前的安全建议和要求。这些机构包括：国际电工技术委员会（IEC）、美国安全试验所（UL）和加拿大标准协会（CSA）。

合规、验证和认证也是认证机构关注的焦点。这些机构包括：德国 TUV 和 VDE（主要面向欧洲），以及 UL 和 CSA（主要面向美国和加拿大市场）。

与安全要求相关的标准的范围十分广泛。其覆盖了许多领域，如：分类、方法、材料、机械、贴标、硬件和软件测试。其目标仅仅是符合可编程电子组件的软件要求，这些要求构成了安全标准的特定部分。在标准的新升级发布时，这些要求很可能已经发生变更。另外，具有共同导向的涉及微控制器通用部件测试（如 *CPU* 或存储器）的安全标准之间具有显著的相似性。

本文档中出现的库基于 ST 开发和应用的测试模块的部分子集，满足 IEC 61508 工业安全标准的严格要求。这些模块经过调整，满足针对居家安全的 IEC 60730 标准。为此，此新库采用了不同于以往发行版本的交付形式。该形式源自工业安全库，目前作为黑盒预编译对象进行交付，无源文件但有清晰的外部接口定义。这一不可变解决方案的优势在于，它完全与工具和配置无关。因此，该解决方案在工具和配置方面完全独立。除此之外，它还独立于 *HAL*、*LL* 或 *CMSIS* 层等其他固件。在用任何更新的编译器版本重新编译先前在更早库版本上验证的源代码文件（普遍做法）时，该解决方案可防止出现意外的编译结果。

表 1. 适用产品

产品编号	订购代码
X-CUBE-CLASSB	X-CUBE-CLASSB

1 概述

1.1 目的和范围

本文档适用于包含 Arm® Cortex®-M33 的 STM32U5 系列 微控制器的专用 X-CUBE-CLASSB 自检库集合。该 X-CUBE-CLASSB 扩展包提供独立于应用程序的软件，以便满足 UL/CSA/IEC 60730-1 安全标准的要求。UL/CSA/IEC 60730-1 安全标准专注于与家用设备和类似电子应用结合使用的电气自动控制装置的安全性。

该软件库的主要目的是促进和加速：

- 用户软件开发
- 受限于相关要求和认证的应用程序的认证过程。

X-CUBE-CLASSB 扩展包在基于 Cortex®-M33 的 STM32U5 系列 微控制器上运行。

arm

提示 *Arm 是 Arm Limited（或其子公司）在美国和/或其他地区的注册商标。*

X-CUBE-CLASSB 扩展包中提供（并与本手册相关）的独立于应用程序的软件测试库（自检库），即 `STL_Lib.a` 文件的版本为 V4.0.0。

1.2 参考文档

[1] UM2875，STM32U5 系列安全手册，适用于面向工业安全的应用程序

[2] AN4435，在任何 STM32 应用中获得 UL/CSA/IEC 60730-1/60335-1 B 类认证的指南，适用于此库的更早版本

2 STM32Cube 概述

2.1 STM32Cube 是什么？

STM32Cube 源自意法半导体，旨在通过减少开发工作量、时间和成本，明显提高设计人员的生产率。STM32Cube 涵盖整个 STM32 产品系列。

STM32Cube 包括：

- 一套用户友好的软件开发工具，覆盖从概念到实现的整个项目开发过程，其中包括：
  - STM32CubeMX 图形软件配置工具 STM32CubeMX，可通过图形向导自动生成初始化 C 代码
  - STM32CubeIDE 一种集外设配置、代码生成、代码编译和调试功能于一体的开发工具
  - STM32CubeProgrammer （STM32CubeProg），图形版本和命令行版本中可用的编程工具
  - STM32CubeMonitor （STM32CubeMonitor、STM32CubeMonPwr、STM32CubeMonRF 和 STM32CubeMonUCPD）功能强大的监控工具，用于实时微调 STM32 应用程序的行为和性能
- STM32Cube MCU 和 MPU 软件包，特定于每个微控制器和微处理器系列的综合嵌入式软件平台（如用于 STM32U5 系列的 STM32CubeU5），其中包含：
  - STM32Cube 硬件抽象层（HAL），确保在 STM32 各个产品之间实现最大限度的可移植性
  - STM32Cube 底层 API，通过硬件提供高度用户控制，确保最佳性能和内存开销
  - 中间件组件的协调集合，如 ThreadX、FileX / LevelX、NetX Duo、USBX、USB-PD、触控库、网络库、 mbed-crypto、TFM 和 OpenBLRTOS、USB 和图形
  - 嵌入式软件实用工具以及全套外设和应用实例
- STM32Cube 扩展包，其中包含嵌入式软件组件，这些组件用以下内容补充 STM32Cube MCU 和 MPU 软件包的功能：
  - 中间件扩展和应用层
  - 在特定的意法半导体开发板上运行的实现案例

2.2 此软件如何补充 STM32Cube？

软件扩展包通过中间件组件扩展 STM32Cube，以便管理特定的基于软件的诊断。

扩展包提供一个通用起点，帮助用户构建和完成应用程序特定的安全解决方案。它包含：

- *STL*：自检库。提供二进制文件和一些源代码，用于管理微控制器通用安全测试的执行。*STL* 是一个独立单元，独立于任何 STM32 软件运行。它集合了微控制器通用组件的自检。
- 用户应用程序：这是一个 *STL* 集成示例，基于一组 STM32Cube 驱动程序，这些驱动程序通过应用程序特定的测试扩展 *STL*。这部分以完整源代码的形式交付，将通过调用额外应用程序特定的模块（由终端用户定义）进行修改或扩展。该示例可用于库测试，包含所有已提供模块的人为故障支持。

3 STL 概述

STL 是 ST 发布的独立于应用程序的软件测试库，旨在针对适用于 STM32U5 系列微控制器的“B 类”相关安全标准要求，实现安全机制的相关子集。*STL* 是独立于 *HAL* / *LL* 的库，专用于 STM32U5 系列微控制器。*STL* 不限定编译工具链，因此任何标准 C 编译器都可对其进行编译。

STL 是自主软件。它按照应用程序的需求执行选定测试，以便检测硬件问题，并将结果报告给应用程序。

STL 部分以对象代码的形式交付（用于库本身），部分以源代码的形式交付，用于用户接口定义和用户参数设置。

3.1 架构概述

STL 测试 Arm® Cortex® M33 *CPU* 内核、Flash 存储器和 *RAM*。

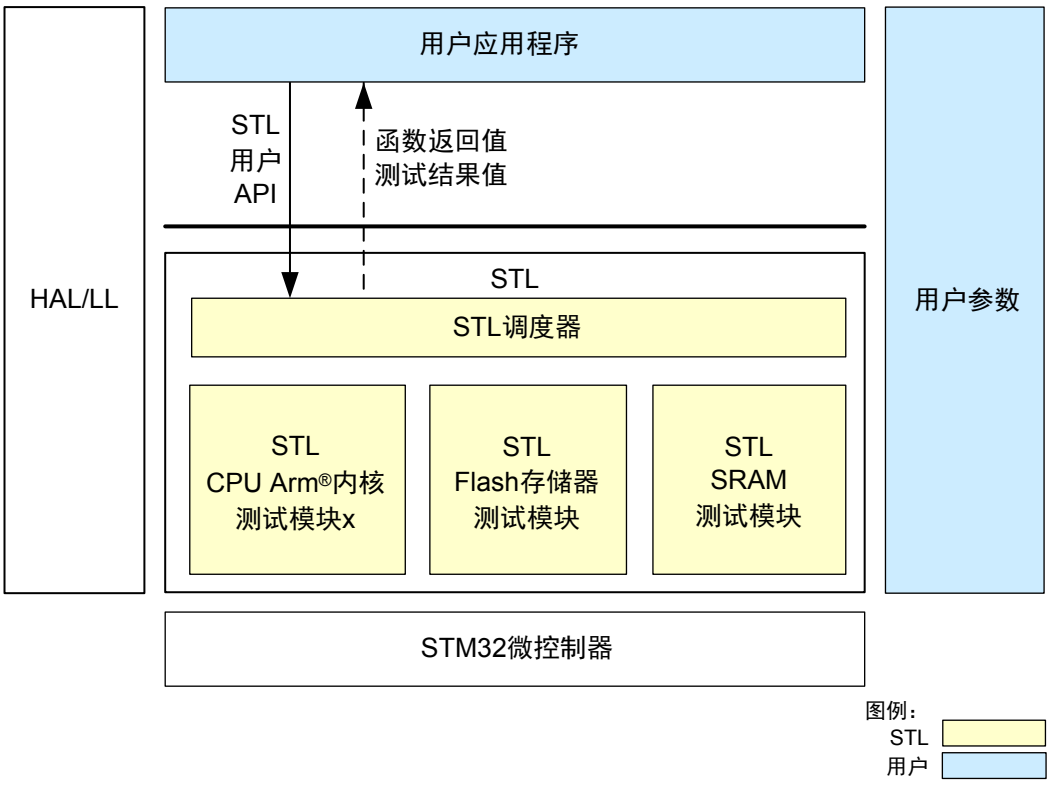
如下图所示，具有集成了 *STL* 的终端用户应用程序的系统架构包含：

- 用户应用程序
- 用户参数
- *STL* 调度器：用户应用程序可直接通过用户 *API* 访问（不是通过 *HAL* / *LL*）
- *STL* 内部测试模块：由 *STL* 调度器调用（对用户应用程序不可见）。

返回到 *API* 层用户应用程序的 *STL* 状态信息（参见表 2）为：

- 函数返回值，收集内部防御性编程检查的结果。
- 测试模块结果值，保存测试结果信息。这些信息部分地对应了模块内部状态（参见第 7.3 节 状态机）。

图 1. *STL* 架构



STL 还允许开发者：

- 使用人为故障功能。开发者可通过强制 *STL* 返回请求的测试结果值来检查应用程序行为。此功能通过特定的用户 API 来提供。

TrustZone®:

当 TrustZone®激活时，必须在安全状态下执行 *STL*。如需从非安全状态执行 *STL*，则必须通过安全网关（*SG*）功能调用。

3.2 支持的产品

STL 运行于以下 STM32 微控制器上：

- STM32U575xx
- STM32U585xx

4 STL 描述

本节介绍关于 *STL* 的功能和性能的基本信息。此外，还总结了终端用户要遵守的限制和强制性操作。

4.1 STL 功能说明

4.1.1 调度器原理

调度器是 *API* 模块，用户应用程序需借此来执行 *STL*。

主调度器：

- 在使用前必须初始化
- 管理：
  - 所用测试模块的初始化和去初始化
  - 所用测试模块的配置
  - 所用测试模块的复位。
- 控制所用测试顺序的执行（*API* 调用）
- 管理对用户调试和集成测试使用的“人为故障”。
- 通过特定的校验和确保重要的内部数据结构的完整性。

调度器控制以下测试的执行：

- *CPU* 测试：在执行任何 *CPU* 测试之前，无需执行任何特定的 *CPU* 测试模块初始化或配置程序（参见第 7.2 节 用户 API 和图 11）。
- Flash 存储器测试作用于 Flash 配置结构的内容之上，这些结构定义了要测试的存储器子集（参见第 7.1 节 用户结构）。这些结构必须由终端用户来填充，并且其内容在 Flash 存储器测试的配置和执行过程中必须保持不变。在执行任何 Flash 存储器测试之前，必须执行 Flash 存储器测试模块初始化和配置程序，参见第 7.2 节 用户 API 和图 12. 状态机图 - Flash 存储器测试 *API*。
- *RAM* 存储器测试作用于 *RAM* 配置结构的内容之上，这些结构定义了要测试的存储器子集（参见第 7.1 节 用户结构）。这些结构必须由终端用户来填充，并且其内容在 *RAM* 测试的配置和执行过程中必须保持不变（在执行 *RAM* 测试之前，必须执行 *RAM* 测试模块初始化和配置程序，参见第 7.2 节 用户 API 和图 13）。

用户通过调度器 *API* 以轮询模式调用 *STL*。可在中断背景下调用 *STL*，但禁止重入。在这种情况下，不能保证 *STL* 的行为。

用户应用程序必须考虑从 *STL* 返回的所有信息，这些信息通过用于采集状态信息的特定预定义数据结构来提供。详细信息见下表。

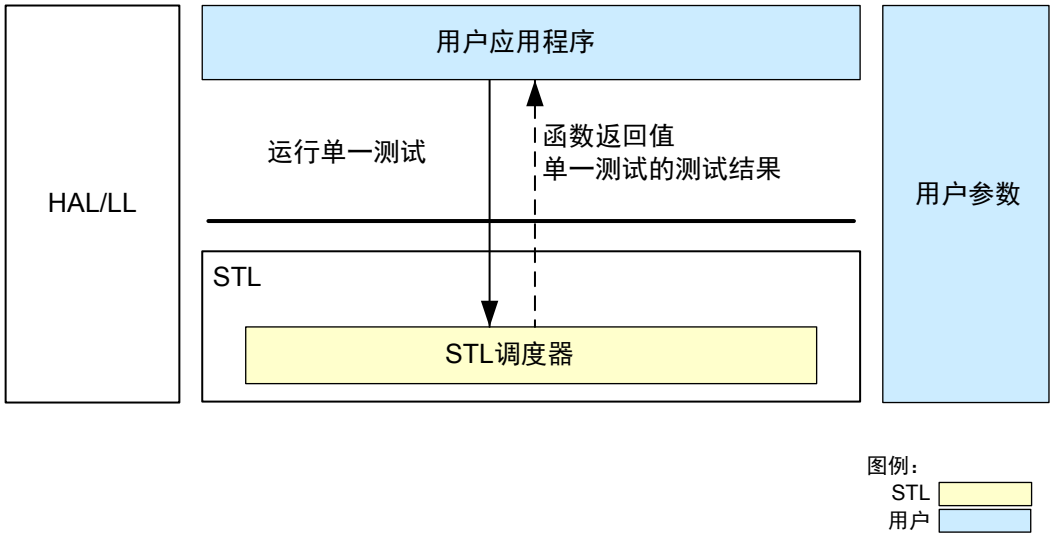
表 2. STL 返回信息

STL 信息	值	说明
函数返回值(1)	STL_OK	调度器函数执行成功
	STL_KO	调度器防御性编程错误（在这种情况下，测试结果无关紧要）
测试模块结果值(2)	STL_PASSED	测试通过
	STL_PARTIAL_PASSED	当测试通过，但尚未到达 *RAM*/Flash 存储器配置的末尾时，仅用于 *RAM* 和 Flash 存储器测试
	STL_FAILED	测试模块执行的硬件错误检测
	STL_NOT_TESTED	未执行测试
	STL_ERROR	测试模块防御性编程错误

1. *请参考第 7.1 节 用户结构中 `STL_Status_t` 的定义。*
2. *请参见第 7.1 节 用户结构中的 `STL_TmSTatus_t`。*

下图中的用户应用程序使用一系列 *API* 调用来处理测试模块执行顺序。

图 2. 单一测试模式架构



可随时中断调度器。一些测试模块可暂时屏蔽中断。有关详细信息，请参见第 4.2.5 节 STL 中断屏蔽时间。

4.1.2 CPU Arm®内核测试

STL 包含下列 *CPU* 测试模块，以及测试功能的概述（仅供参考）：

- TM1L：实现通用寄存器的轻量测试模式
- TM7：实现两个栈指针（*MSP* 和 PSP）的模式和功能测试 *PSP*
- TMCB：实现 *APSR* 状态寄存器的测试。

Caution: *STL CPU* 测试被分割成单独的测试模块。这样做不是为了允许全部可用 *CPU TM* 的部分执行，而是作为一项支持功能，在终端用户应用程序中实现更好的 *CPU* 测试调度，如时间限制。在默认情况下，假定要执行所有可用的 *TM*。

4.1.3 Flash 存储器测试

原理

Flash 存储器测试涉及 STM32U5 系列的嵌入式 Flash 存储器。

为了提供正确的 Flash 存储器测试配置，必须具备以下结构。

- 块：4 字节（FLASH_BLOCK_SIZE）的连续区域，由 *STL* 进行硬编码。
- 分区：1024 字节（FLASH_SECTION_SIZE）的连续区域，由 *STL* 进行硬编码。与 Flash 存储器物理扇区无关。Flash 存储器分为多个分区。第一个分区始于 Flash 存储器的第一个地址，后面的分区彼此相接。

  用户必须确保每个分区的 *CRC* 校验和的正确计算和安置，将在 Flash 存储器完整性测试期间进行检查。
- 二进制文件（图 4 中称为“用户程序”）：编译器提供的连续代码区。它始于一个分区的开头。当二进制文件区的大小不是分区大小的倍数时，通常以一个不完整的分区结束。在任何情况下，二进制文件都必须 32 位对齐（参见以下 ST *CRC* 工具信息）。当 TrustZone®激活时，通常会创建两个二进制文件：一个安全二进制文件和一个非安全二进制文件。

- 子集：用户定义的分区组成的连续区域。用户应用程序可定义一个或多个子集。子集必须定义在一个二进制文件区以内。其起始地址必须与分区的开头对齐。它只能包含具有对应的预计算 *CRC* 值的分区。当子集的最后一个分区是二进制文件的最后一个部分时，分区可能不完整。用户应用程序必须将子集末尾与二进制文件区的结束地址对齐。如果单独测试一组完整分区，则子集结束地址必须与测试的最后一个分区的末尾对齐。

  子集的计算如下：

  子集大小 = K * FLASH_SECTION_SIZE + L * FLASH_BLOCK_SIZE

  其中：

  – K 为大于 0 的整数。
  – 0 ≤ L < (FLASH_SECTION_SIZE / FLASH_BLOCK_SIZE) 当 L > 0 时，二进制文件的最后一个分区不完整。

  由用户应用程序定义子集。

提示 *可以定义多个子集。*

STL 利用以下原则实现 Flash 存储器的测试（基于用户配置结构的实际内容）：

- 在用户应用程序定义的一个或多个子集的分区上执行测试。
- 针对用户应用程序定义的分区数，连续（一次性）执行或在单个原子步骤中部分执行测试。
- 测试结果基于计算所得 *CRC* 值（在测试执行过程中计算）与预期 *CRC* 值（在软件二进制文件刷写前计算）之间的 *CRC* 比较。

执行 Flash 存储器测试的必要步骤（对用户应用程序而言）：

- 测试初始化
- 配置一个或多个子集
- 执行测试。

在测试了所有子集后，用户需要将 Flash 存储器测试模块复位，以便再次执行测试。

如果测试结果为 STL_ERROR / STL_FAILED，则测试模块将卡在失败的存储器子集处。在这种情况下，去初始化、初始化并重新配置 Flash 存储器，然后再次运行测试。

预期 CRC 的预计算

Flash 存储器测试基于内置的硬件 *CRC* 单元或软件 *CRC*，可通过标记进行配置。默认配置是使用硬件 *CRC*。如需使用软件 *CRC*，则必须按照第 5.5.2 节 从头开始构建应用程序的步骤中步骤 3 的描述使能标记 STL_SW_CRC。*CRC* 是符合 IEEE 802.3 标准的 32 位 *CRC*。

预留了 Flash 存储器的一部分作为 *CRC* 专用区，其大小取决于 Flash 存储器的大小。该区域采用字段格式，其中的每个 Flash 存储器分区都预留了足够空间用于存储 32 位 *CRC* 方案。用户必须为要测试的所有分区计算出有效 *CRC* 方案并将其存储在字段中。相关内容如图 3 所示。

当 TrustZone®激活时，Flash 存储器中必须有与测试的子集相对应的 *CRC* 区。如果在非安全 Flash 存储器和安全 Flash 存储器中都定义了要测试的子集，则必须有一个非安全 Flash 存储器专用的 *CRC* 区和一个安全 Flash 存储器专用的 *CRC* 区。

从二进制文件开始到结束，为二进制文件的每个连续分区预计算一个预期 *CRC* 值。这意味着可测试分区的数量取决于二进制文件大小。二进制文件区与分区通常未进行对齐。在这种情况下，预计算最后一个不完整分区的 *CRC* 校验值并在覆盖了二进制文件区的分区部分进行单独测试。

先决条件：

- 用户程序区必须始于一个分区的开头
- 用户程序区的边界必须是 32 位对齐。
- 根据 Flash 存储器总体大小和用户程序大小，最后一个程序数据和第一个 *CRC* 数据可能保存在同一个 Flash 存储器分区（无任何重叠）。在这种情况下，只能基于用户程序数据计算 *CRC*，参见图 4 中的示例 3。

ST CRC 工具信息

ST 提供 *CRC* 预计算工具。该工具作为 STM32CubeProgrammer 内部的单个功能来提供（参见第 6.2.2 节 工具设置），会自动用填充位（0x00 模式）填充二进制文件，以便实现 32 位对齐。

图 3. **Flash 存储器测试：*CRC* 原理**

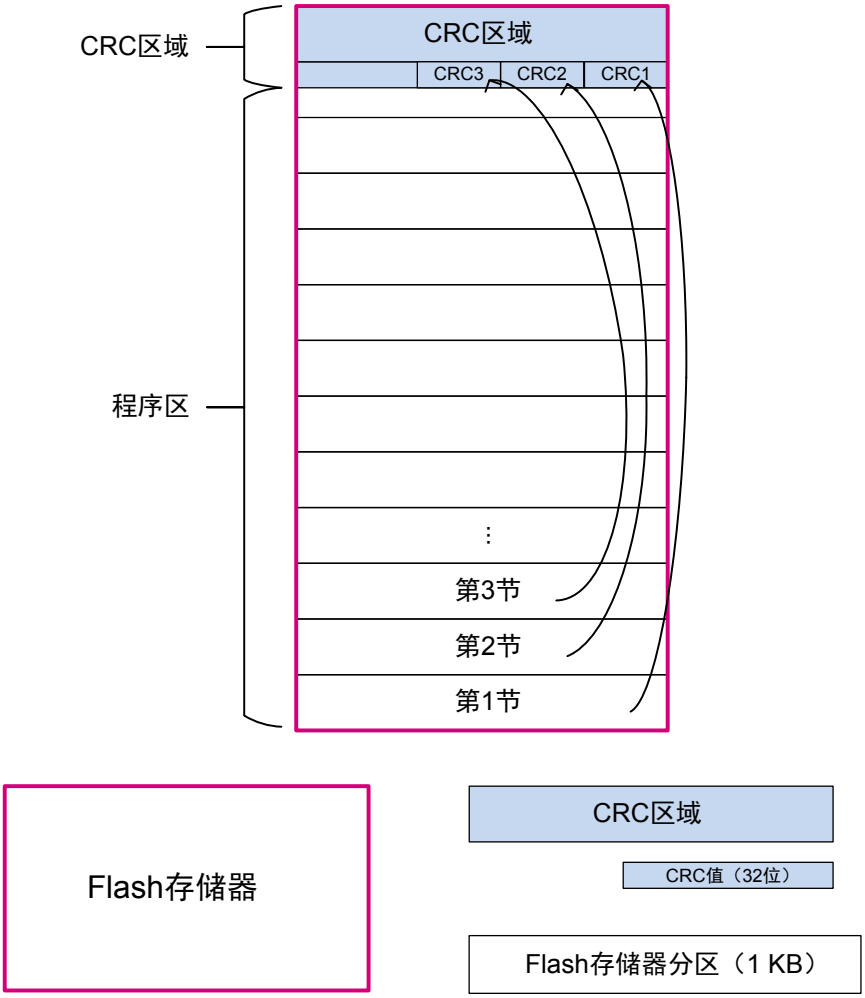


图 4. **Flash 存储器测试：*CRC* 用例 vs 程序区**

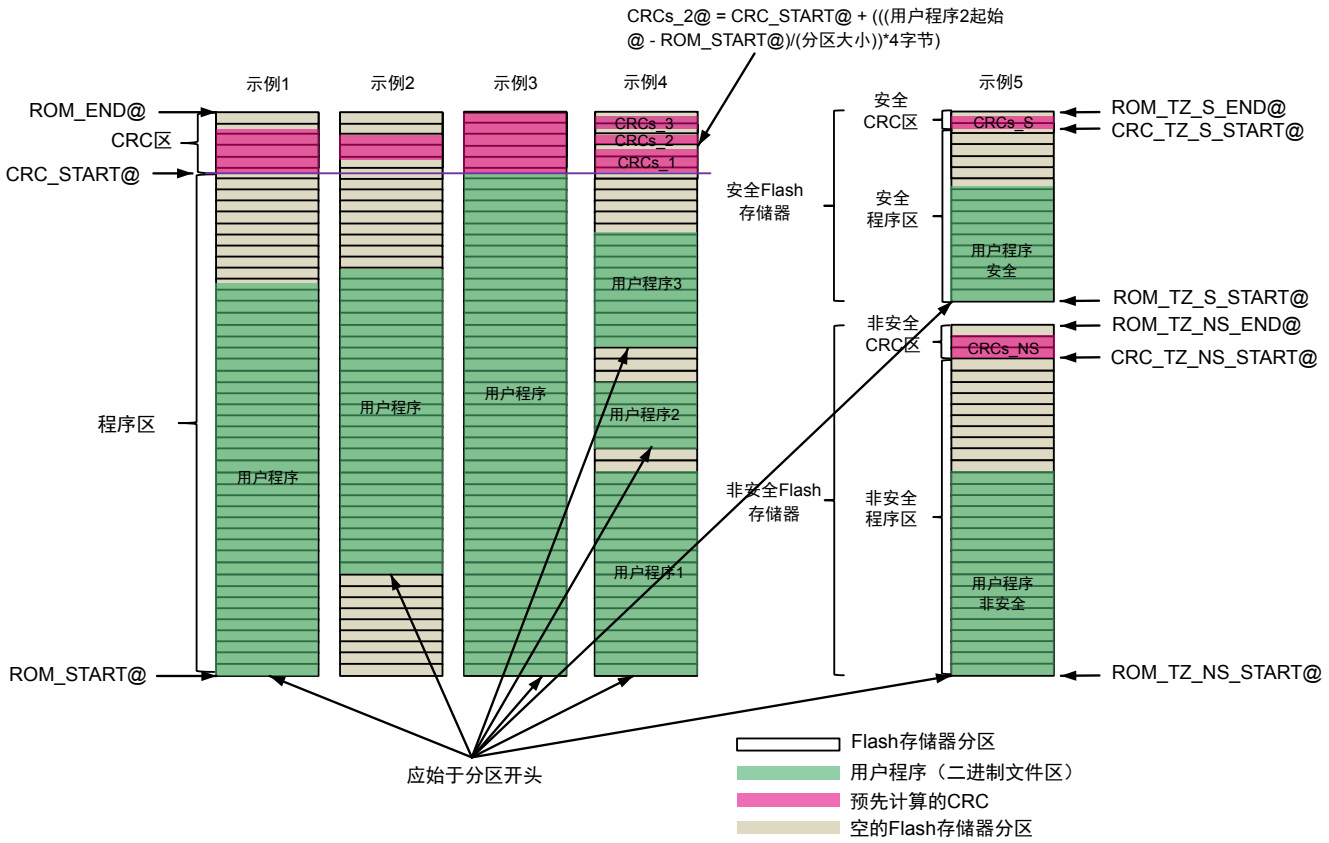


图 4 所示用例的描述：

- 示例 1：用户程序始于 ROM_START 地址，因此从 CRC_START 地址开始保存 *CRC*。
- 示例 2：用户程序始于一个分区的开头而非 ROM_START。保存的 *CRC* 始于 *CRC* 区的合适地址。
- 示例 3：用户程序使用完整程序区，因此最后一个程序数据与第一个 *CRC* 数据保存在同一个 Flash 存储器分区（无任何重叠）。
- 示例 4：在三个分开的区中定义用户程序。这需要三个分开的 *CRC* 数据区。
- 示例 5：激活了 TrustZone®。Flash 存储器中的用户程序被划分成一个具有专用 *CRC* 区的安全二进制文件和一个具有专用 *CRC* 区的非安全二进制文件。

CRC 起始地址计算：

- 实际计算：

  CRC_START 地址 = (uint32_t *)(ROM_END - 4 * (ROM_END + 1 - ROM_START)/(FLASH_SECTION_SIZE) + 1)；其中，FLASH_SECTION_SIZE = 1024
- 文本转换：

  CRC_START = ROM_END - (以字节为单位的 *CRC* 大小) * (Flash 存储器分区数) + 1

Flash 存储器测试和中断

可随时中断 Flash 存储器 *TM*。

4.1.4 RAM 测试

原理

RAM 测试涉及 STM32U5 系列的嵌入式 SRAM 存储器。

为了提供正确的 *RAM* 测试配置，必须具备以下结构。

- 块：16 字节（RAM_BLOCK_SIZE）的连续区域，由 *STL* 进行硬编码（与存储器物理扇区无关）。
- 分区：128 字节（RAM_SECTION_SIZE）的连续区域，由 *STL* 进行硬编码。
- 子集：一个连续区域，大小为两个块的倍数，具有 32 位对齐的起始地址。由于子集的最后一个部分可能小于一个分区，因此子集大小不一定是分区大小的倍数。
- 子集大小 = N * RAM_SECTION_SIZE + 2 * M * RAM_BLOCK_SIZE，

  其中：
  - N 为 ≥ 0 的整数
  - M 为 0 ≤ M < 4 的整数，当 M > 0 时，子集最后部分的大小与分区大小不一致。

  由用户应用程序定义子集。

提示 *可以定义多个子集。*

STL 利用以下原则实现 *RAM* 存储器测试（基于用户配置结构的实际内容）：

- *RAM* 由用户应用在 *RAM* 块上执行测试
- *RAM* 针对用户应用程序定义的分区数，连续（一次性）执行或在单个原子步骤中部分执行测试
- 测试实现基于 March C-算法

执行 *RAM* 测试的必要步骤（对用户应用程序而言）：

- 初始化 *RAM* 测试
- 配置一个或多个 *RAM* 子集
- 执行 *RAM* 测试

在测试了所有子集后，应用必须将 *RAM* 测试模块复位，以便再次执行测试。

如果测试结果为 STL_ERROR / STL_FAILED，测试模块将卡在失败的存储器子集处。在这种情况下，去初始化、初始化并重新配置 *RAM*，然后再次运行测试。

RAM 测试和中断

可随时中断 *RAM TM*。*RAM TM* 在最小数据粒度时间内屏蔽中断（参见第 4.2.5 节 STL 中断屏蔽时间）。

除非用户应用程序激活了专用 STL_ENABLE_IT 编译开关，否则默认在最小数据粒度（块）执行期间暂时屏蔽 STM32 中断和优先级可配置的 Cortex®异常（参见第 5.5.2 节 从头开始构建应用程序的步骤）。如果标记激活，则由终端用户负责管理 *STL* 与应用软件之间的干扰。这些接口可能导致虚假的 STL 错误报告或应用软件故障，起因是测试导致的 *RAM* 内容损坏（参见 March C-测试原理）。

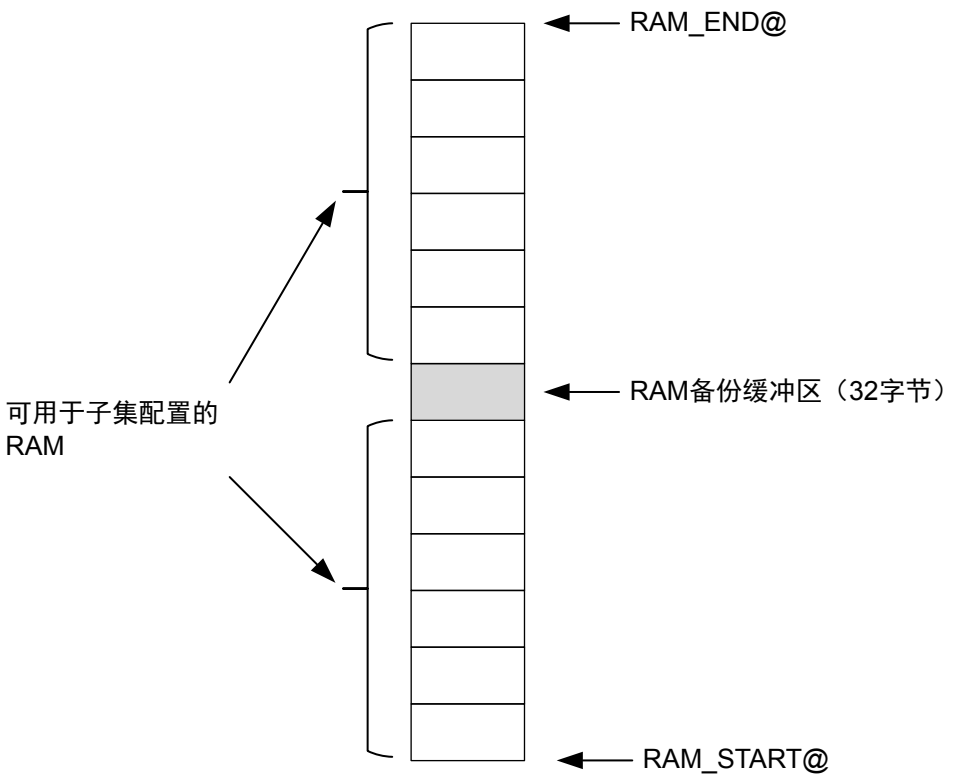
March C-测试原理

RAM 测试基于 March C-算法，存储器以特定模式写入，然后回读。使用备份缓冲区恢复存储器初始内容。*RAM* 备份缓冲区的地址可由终端用户在编译时配置。它必须在 *RAM* 子集配置之外。整个测试只有一个 *RAM* 备份缓冲区。March C-测试还测试 *RAM* 备份缓冲区。

提示 *当 TrustZone®激活时，备份缓冲区必须位于安全 RAM 中。*

出于安全原因，在 STM32U5 系列微控制器中必须使用备份缓冲区，而在其他 STM32 器件中为选用。这意味着选项 STL_DISABLE_RAM_BCKUP_BUF 不可用且无效。

图 5. **RAM 测试：使用情况**



4.2 STL 性能数据

使用以下测试设置获得数据：

- *STL* 应用程序：编译过程中提供了 STL 库的编译详情。
- 性能测试项目使用 Arm®（EWARM）工具链 v9.20.1 的 IAR 嵌入式工作台®进行编译
- 在以下条件下编译了软件配置：
  - HCLK 时钟设置为 160 MHz
  - Flash 存储器延迟设置为 4 个等待状态
  - *ICache* 激活
  - NUCLEO-U575ZI-Q 版本 B（MB1549 C）
  - *STL* 在安全二进制文件中运行，由在非安全二进制文件中运行的用户应用程序调用

4.2.1 STL 执行时间

表 3 汇总了采用最佳默认 *STL* 设置时的 *STL* 执行时间。第 8 节 STL：执行时间详情中详细列出了每个 *API* 的测量值。

表 3. **STL 执行时间，时钟频率：160 MHz**

测试的模块	条件		结果（单位：时钟周期）
CPU	TM1L、TM7、 TMCB		4994
Flash 存储器	默认配置（不使能 STL_SW_CRC）：	在安全 Flash 存储器中测试了 1 千字节	2981
		在安全 Flash 存储器中测试了 10 千字节并在非安全 Flash 存储器中测试了 1 千字节	25713
	STL_SW_CRC 使能：	在安全 Flash 存储器中测试了 1 千字节	12524
		在安全 Flash 存储器中测试了 10 千字节并在非安全 Flash 存储器中测试了 1 千字节	130765
RAM	默认配置（不使能 STL_ENABLE_IT）：	在安全 RAM 中测试了 128 字节 *RAM*	5336
		在安全 *RAM* 中测试了 272 千字节并在非安全 RAM 中测试了 512 千字节 *RAM*	27353568

4.2.2 STL 代码长度和数据量

STL 代码长度和数据量如下表所示。

表 4. **STL 代码长度和数据量（单位：字节）**

配置	模块	Flash 存储器代码	Flash 存储器 RO 数据	R/W 数据
不使能 STL_SW_CRC	`stl_user_param_template.o`	-	12	56
	`stl_util.o`	268(1)	-	8
	`stl_lib.a`	5276	1449	184
STL_SW_CRC 使能	`stl_user_param_template.o`	-	12	56
	`stl_util.o`	124(1)	-	4
	`stl_lib.a`	5276	1449	184

1. *当使用软件 CRC 计算时，编译的代码/函数较少（少了 CRC 硬件初始化），因此对象代码量相较于硬件 CRC 有所减少。*

4.2.3 STL 栈用量

STL 执行可用 *API* 所需的最小可用栈空间为 200 字节。

4.2.4 STL 堆用量

STL 从不使用动态分配，因此堆用量与 *STL* 无关。

4.2.5 STL 中断屏蔽时间

在 *CPU* TM7 和 *RAM* 测试期间，*STL* 多次屏蔽 STM32 中断和优先级可配置的 Cortex®异常。下表所示为 *RAM* 测试的最长中断时间（参见表 5）。

表 5. *STL* 最长中断屏蔽信息

测试的模块	持续时间（最大值），单位：时钟周期	注释
RAM	471	在测试的部分步骤中，每次执行 STL_SCH_RunRamTM 函数都会在以下时间段执行一系列中断屏蔽： • 备份缓冲区：471 个时钟周期+ • 要测试的第一个 *RAM* 块：386 个时钟周期 • 要测试的每个中间 *RAM* 块：462 个时钟周期(1) • 要测试的最后一个 RAM 块：387 个时钟周期
CPU TM7	422	屏蔽两次，分别是 422 个和 372 个时钟周期

1. *每次执行 RAM 测试涉及的 RAM 块数量（要求是两个 RAM_BLOCK_SIZE 的倍数）取决于用户结构的内容（已定义子集的大小 vs 原子步骤 - 参见第 4.1.4 节 RAM 测试）*

4.3 STL 用户限制

终端用户需考虑应用程序与 *STL* 之间的干扰。忽略这一点可能导致虚假的 *STL* 错误报告和/或应用软件执行问题。相应地，为防止出现任何干扰，应用软件和 *STL* 集成必须符合本节列出的每项限制要求。

4.3.1 特权级

STL 必须以特权级执行，以便能够修改特定的内核寄存器（如 PRIMASK 寄存器）。
如果特权级未设置为特权，*CPU* TM7 将返回 STL_ERROR。

4.3.2 RCC 资源

在 *STL* 执行期间，RCC 被配置为在所有 TM 执行期间为 *CRC* 提供计时。这意味着：

- 当 *STL* 返回时，它为 CRC 恢复用户 RCC 时钟设置（使能或禁用） *CRC*
- 在 *STL* 执行期间通过保存/恢复 *STL* 设置配置 RCC 时，用户应用程序应小心谨慎。

4.3.3 CRC 资源

若使用硬件 *CRC*，则 *STL* 依赖于 STM32 CRC IP。在 *STL* 执行期间使用 *CRC* 资源的情况有两种：

- 在执行 *STL* 初始化（函数 STL_SCH_Init）的过程中：使用硬件 *CRC*。应用软件不能修改该阶段硬件 *CRC* 的使用，因此，在 STL_SCH_Init 函数的执行过程中，STL_SW_CRC 标记无影响。
- 在其他 STL 函数的执行过程中：应用程序可通过 STL_SW_CRC 标记在硬件 *CRC* 和软件 *CRC* 之间进行选择。默认使用硬件 *CRC*（禁用 STL_SW_CRC 标记）。

使用硬件 *CRC* 意味着：

- 在调用 *STL* 之前，用户应用程序必须保存完整的硬件 *CRC* 配置。在 *STL* 执行之后，必须恢复用户配置。
- 在 *STL* 执行期间，硬件 *CRC* 被配置并用于满足 *STL* 的需求（在 *STL* 执行期间使用 *CRC* 时，用户应用程序必须保存/恢复 *STL* 设置）。

4.3.4 APSR 的 Q 位

CPU TMCB 执行使 *APSR* 的 Q 位（粘着位）置位。用户应用程序在使用 Q 位时必须考虑这一点。

4.3.5 中断管理

上报机制 - Arm® Cortex® 行为提示器

当 *STL* 禁用 STM32 中断和优先级可配置的 Cortex®异常时，注意 Arm® Cortex®可能会上报 HardFault。在这种情况下，调用 HardFault 处理程序而不是故障处理程序。

中断和 CPU TM7

除非用户应用程序激活了 STL_ENABLE_IT，否则默认在 CPU TM7 期间屏蔽 STM32 中断和优先级可配置的 Cortex®异常（参见第 5.5.2 节 从头开始构建应用程序的步骤）。

如果激活了 STL_ENABLE_IT 标记，由于修改了流水线顺序，将无法保证 STL CPU TM7 行为的正确性。这可能导致 *STL* 生成虚假的测试错误报告或不检测硬件故障。

中断 *RAM* March C 测试

除非用户应用程序激活了 STL_ENABLE_IT，否则默认在 *RAM* March C-测试期间屏蔽 STM32 中断和优先级可配置的 Cortex®异常（参见第 5.5.2 节 从头开始构建应用程序的步骤）。

如果激活了 STL_ENABLE_IT 标记：

- 由于应用程序在其中断处理期间可能会覆盖 STL 测试的 *RAM* 内容，因此无法保证 *STL RAM* 测试行为的正确性。这可能导致 *STL* 生成虚假的 *RAM* 测试错误报告。
- 用户应用软件的行为可能受到影响。可能从 *STL* March C-测试正在修改的 *RAM* 位置读取或使用错误数据。

4.3.6 *STL* 如何屏蔽中断

为了屏蔽中断，*STL* 将 PRIMASK 寄存器的 PM 位字段置为 1：

- 在无安全扩展（TrustZone®禁用）的实现中，将此位置为 1 会促使当前执行优先级变为 0，将屏蔽所有优先级可编程的异常。
- 在有安全扩展（TrustZone®使能）的实现中，在安全状态下执行 *STL*。将 PRIMASK_S 寄存器的 PM 位字段置为 1 会促使当前执行优先级变为 0，将屏蔽所有优先级可编程的异常。

因此，当当前执行优先级变为特定值时，将屏蔽所有具有更低或相等优先级的异常。

异常屏蔽机制基于异常优先级而非异常状态（安全或非安全）。

提示 *AIRCR 寄存器 PRIS 位字段置为 1 会使非安全异常的优先级编码少一位，非安全异常的最高优先级 = 0x80。*

如果 AIRCR 寄存器 PRIS 位字段为：

- *0，将 PRIMASK_NS 置为 1 会促使当前执行优先级变为 0x0 => 屏蔽所有异常（复位、NMI 和 Hardfault 除外）*
- *1，将 PRIMASK_NS 置为 1 会促使当前执行优先级变为 0x80 => 只屏蔽 NS 异常*

4.3.7 DMA

应用程序必须管理 DMA，以避免 *STL* March C-测试期间对 *RAM* 存储区的意外访问。在这种情况下：

- DMA 写操作可能干扰 *STL* 测试，导致虚假错误报告
- *STL* 覆盖 DMA 专用 *RAM* 分区可能导致 DMA 读操作返回错误数据。

4.3.8 存储器映射

由于 *RAM* 测试模块和 March C 算法设计，用户必须确保 *STL* 的“只读”数据位于 Flash 存储器中。这必须通过适当修改相关的链接器文件来实现。

下面以 EWARM 和 STM32CubeIDE 为例。

EWARM .icf 文件修改示例

```
place in ROM_region { readonly };
```

STM32CubeIDE.ld 文件修改示例

```
.rodata :
{
……
} >ROM
```

提示 *默认配置通常将“只读”数据放在 Flash 存储器中。*

4.3.9 处理器模式

STL 必须以线程模式执行，以便将激活的栈指针设置为进程栈指针。

如果 *STL* 不以线程模式执行，*CPU* TM7 将返回 STL_ERROR。

4.3.10 TrustZone®

当 TrustZone®激活时，必须将 *STL* 库一次性嵌入安全二进制文件中。如需从非安全状态执行 *STL*，则必须通过 *SG*（安全网关）功能调用。

4.4 终端用户集成测试

本节将介绍在验证阶段终端用户要执行的必要测试。这些测试能保证将 *STL* 正确集成到应用软件中。

4.4.1 测试 1：正确的 STL 执行

终端用户必须使用预期的函数返回值和预期的测试模块结果值（参见第 7.2 节 用户 API），检查计划的每个诊断功能是否正确执行。这涉及测试模块执行及其所有配置操作。

4.4.2 测试 2：正确的 STL 错误消息处理

终端用户必须检查 *STL* 函数返回是否生成了任何错误信息，以及测试模块结果值是否被正确解读为不符合预期的行为并在其应用软件中得到正确处理。错误信息指不同于预期值的值，参见第 7.2 节 用户 API）。在验证过程中，对于使用的每项功能，必须使用人为故障功能，枚举与关联的各项软件诊断（*CPU* 测试、*RAM* 测试、Flash 存储器测试）相关的错误测试模块结果值的生成。

不能将此过程视为真实器件上实际 *CPU* 故障的详尽模拟，而应是所实现 *API* 的测试接口的模拟。

5 软件包说明

本节将详细介绍 X-CUBE-CLASSB 扩展包的内容及其正确使用方法。

5.1 概述

X-CUBE-CLASSB 是 STM32U5 系列 微控制器的软件扩展包。
它提供完整解决方案，帮助终端用户构建安全的应用程序：

- 提供独立于应用程序的软件测试库：
  - 部分以对象代码的形式：`STL_Lib.a`，即库本身
  - 部分以源文件的形式：`stl_user_param_template.c` 和 `stl_util.c`
  - 有三个头文件：`stl_stm32_hw_config.h`、`stl_user_api.h` 和 `stl_util.h`
- 以源代码的形式提供一个用户应用程序示例。

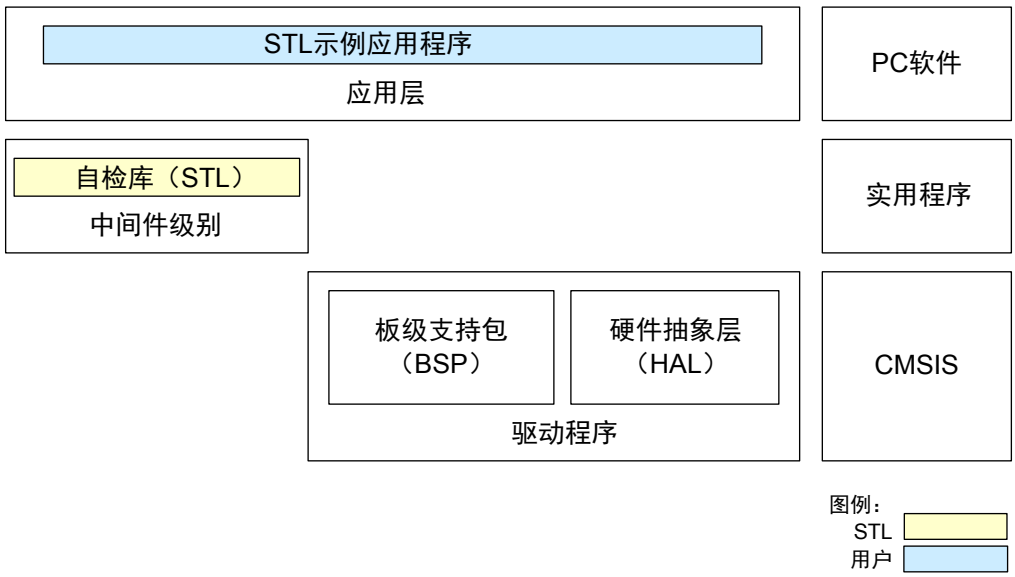
X-CUBE-CLASSB 已移植到第 3.2 节 支持的产品中列出的产品上。
软件扩展包包含了应用示例，开发人员可以将其用于试验代码。它以 zip 存档文件的形式提供，包含源代码和库。
可支持以下集成开发环境：

- IAR 嵌入式工作台® Arm®（EWARM）
- Keil® 微控制器开发套件（MDK-ARM）
- STM32CubeIDE.

5.2 架构

图 1 描述了 X-CUBE-CLASSB 扩展包的组成部分。

图 6. 软件架构概述



5.2.1 STM32Cube HAL

HAL 驱动层提供简单、通用的多实例 *API* 集合（应用程序编程接口），以便与上层（应用程序、库和栈）交互。
它包括通用和执行 *API*。它直接围绕通用架构构建，允许在其基础上的软件层，例如中间件层，实现了它的功能又无需依赖给定微控制器的特定硬件配置。
这种结构可提高库代码的可复用性，并确保可向其他设备轻松移植。

5.2.2 板级支持包（BSP）

除 MCU 之外，软件包需支持 STM32 板上的外设。板级支持包（BSP）中包含此软件。这是一个有限的 *API* 集合，为一些特定的板组件（如 LED 和用户按钮）提供编程接口。

5.2.3 STL

STL 的重要部分，在中间件层提供，是管理基于软件的诊断测试的黑盒。它独立于 *HAL*、*BSP* 和 *CMSIS*，即使是 *STL* 集成示例也依赖于某些 *HAL* 驱动程序。

5.2.4 用户应用程序示例

示例展示了如何将可能的 *STL* 测试模块调用序列集成到应用程序中，确认 *API* 的返回并人工模仿其故障响应。另外，还包含一个用于测试时钟系统的特殊模块，它采用符合“*B 类*”标准要求的监测方法，具有完整源代码，便于扩展可用库集合。它演示了如何通过特定的测试或模块来扩展库，这些模块完全由终端用户定义。

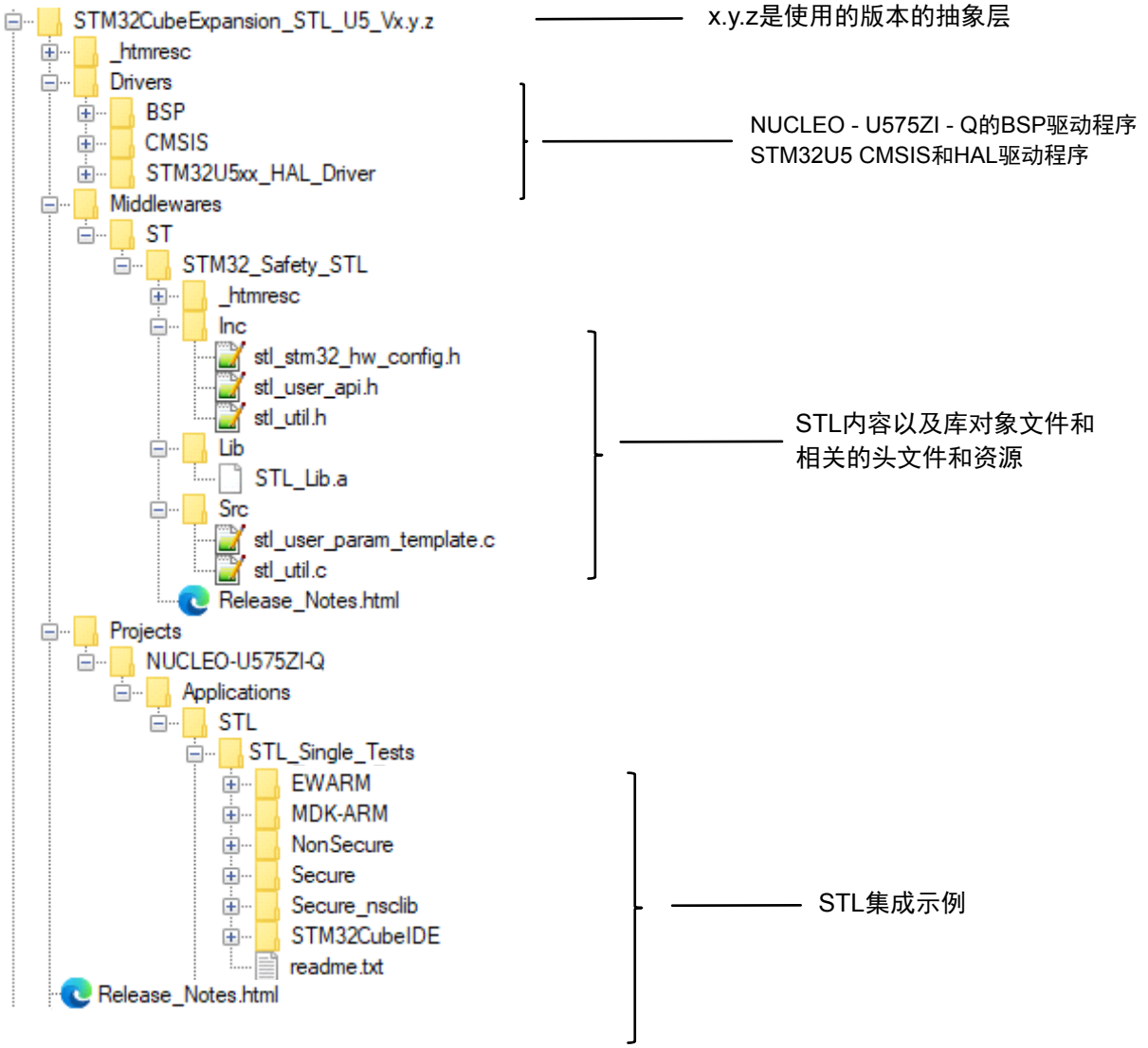
5.2.5 STL 完整性

通过哈希 SHA-256 确保 STL 内容的完整性。

5.3 文件夹结构

图 7 中显示了结构的顶层视图。

图 7. 项目文件结构



5.4 API

5.4.1 合规

接口合规

STL 的库部分不在源代码中交付。此库使用 IAR 嵌入式工作台® for Arm v9.20.1 进行编译。为了满足 EWARM 帮助部分的“AEABI 合规”中描述的 AEABI 合规要求，使用`--aeabi` 和`--guard_calls` 编译选项执行编译。

为了符合 IEC 61508 功能安全标准，使用经安全认证的 EWARMFS v8.50.1 编译器编译了经 TUV 认证的 *STL* 工业版本。此 *STL* 的源文件基于库的工业版本，为符合 IEC 60730-1 标准进行了调整，并通过 EWARM 编译器的最新可用版本进行了重新编译。该 EWARM 编译器版本未经过专门的安全认证。但是，至少软件开发要求提取自 IEC 61508-3。

安全指南

为了满足 IAR 嵌入式工作台®安全指南（建议 2.1-1、2.2-5、2.4-1a 和 5.4-3）和 Keil®安全手册（§4.9.2）中描述的安全准则合规要求，通过`--strict`、`--remarks`、`--require_prototypes` 和`--no_unaligned_access` 编译选项实现合规。

库合规

STL 的库部分（不以源代码交付）符合 C 标准库 ISO C99 的要求。

它已通过 IAR™选项进行了编译。C 语言方言 = C99。

Arm 编译器 C 工具链供应商/版本独立性

STL 用户 *API* 仅指 C 语言“`uint32_t`”和“`enum`”类型：

- C 语言“`uint32_t`”类型具有 32 位的固定类型大小（依据 C 标准 C99）
- C 语言“`enum`”类型大小（依据 C 标准 C99）通过实现定义。它必须能够代表所有枚举成员的值。在 *STL* 接口中，`enum` 类型值是无符号整数，小于或等于（2^{32} - 1）。用户必须确保 `enum` 类型值可保存 32 位值。

5.4.2 依赖关系

STL 库调用 `memset` 标准 C 库函数。

此外，IAR™ EWARM 工具链编译器被用于编译 *STL* 库。在某些情况下，此编译器可调用以下标准 C 库函数：`memcpy`、`memset` 和 `memclr`。此行为是 IAR™ EWARM 工具链编译器固有的，不能禁用或避开。

因此，在链接 *STL* 库时，用户必须确保这些标准 C 库函数得到定义。用户可以使用工具链提供的函数或用户自有的函数。

5.4.3 详细信息

关于可用 *API* 的详细技术资料可在第 7.2 节 用户 API中找到，其中描述了函数和参数。

5.5 应用程序：编译过程

5.5.1 交付的 STL 示例的构建步骤

在交付的 *STL* 示例中，*STL* 在安全二进制文件中运行，应用程序在非安全二进制文件中运行。务必完成以下步骤：

1. 安装 ST *CRC* 工具（参见第 6.2.2 节 工具设置）或其他 *CRC* 工具，它将生成正确执行 Flash 存储器测试所需的适当结构。
2. 项目选择。选择并打开一个项目示例。

提示 *关于后续步骤，请参考发布包中提供的示例的 readme.txt 文件。根据 IDE，安全和非安全二进制文件的管理有些许差异。*

3. 执行。

启动板件并检查结果：

- LED 正常亮灭：测试结果符合预期。
- LED 不正常亮灭：存在错误。

如果任何测试返回失败结果，则 LED 每 2 秒闪烁一次。如果 *STL* 检测到防御性编程错误，则 LED 每 4 秒闪烁一次。

然后，调用 `FailSafe_Handler` 程序，用参数保存失败模块的识别码

提示 *`stl_user_api.h` 文件中给出了代码定义，如果发生防御性编程错误，会在模块代码中添加 DEF_PROG_OFFSET。*

提示 *在编译/链接二进制文件后，CRC 工具执行两项操作：*

- *计算 CRC（如果发生错误，检查 CRC 工具路径）。详情请见 第 5.5.2 节 从头开始构建应用程序的步骤*
- *自动将计算所得 CRC 添加到生成的二进制文件。*

5.5.2 从头开始构建应用程序的步骤

为了从头开始构建应用程序，请按照下列步骤操作：

1. 使用合适的目录结构和所有合适的文件包新建应用程序项目。使用 STM32CubeMX 工具自动完成此过程。
2. 如果 STM32CubeMX 工具不支持项目中 *STL* 的任何自动化包含选项，则从交付的 *STL* 示例中将…`Middleware\ST\STM32_Safety_STL` 目录下的内容粘贴并复制到应用程序项目目录结构中。请参见第 5.3 节 文件夹结构。在这种情况下，在执行后续步骤的同时修改项目设置：
   - 将 SCR 目录下的所有 *STL* 源文件添加到项目
   - 将 INC 目录指定为项目的附加目录
   - 强制链接器将 LIB 目录下的库对象文件作为附加库包含在内

提示 *仅在 CubeMX 工具不支持任何自动包含选项时才需要执行上述步骤，否则将完全由工具来执行 - 无需任何人工干预（如前文所述）- 用户可以忽略并继续执行步骤 3*

3. 如果需要，在项目设置中添加以下可选择的预处理器编译开关：
   - 使能 STL_SW_CRC 的选项：用户应用程序将在此处选择软件 *CRC*。如未激活，则默认使用硬件 *CRC* 计算。
   - 使能 STL_ENABLE_IT 的选项：用户应用程序将在此处使能 CPU TM7 和 *RAM* 测试期间的 STM32 中断。如未激活，则将在这些测试期间屏蔽中断。见第 4.3.5 节 中断管理和第 4.1.4 节 RAM 测试。
4. 检查 Flash 存储器密度配置。

   必须在 *stl_user_param_template.c* 文件中为项目的 Flash 存储器设置正确范围。特别是应更新 STL_ROM_END_ADDR，以确保与相关的链接器分散加载描述文件和 *CRC* 工具脚本一致（参见步骤 6.）。
5. 开发用户 *STL* 流程控制。按照定义的安全任务的要求，通过实现按周期重复的合适 *API* 调用顺序来完成。

   必须确保所有相关用户结构的正确填充，以便控制存储器测试并应用正确的 *STL* 返回信息检查。请参见第 7 节 STL：用户 API 和状态机。
6. 应用 *CRC* 工具，构建 *CRC* 计算所必需的 *CRC* 区内容。请参见第 6.2.2 节 工具设置。

   执行 STM32CubeProgrammer 的合适命令行。通过在编译过程中触发 IDE 构建后功能操作，可自动完成此操作，如图 8 和图 9 所示。
7. 编译、加载并执行二进制文件。

如果 *STL* 检测到硬件故障，则可使用人为故障 *API* 调试已编程 *STL* 流程的正确行为。

提示 *当 TrustZone® 激活时，STL 必须内置于安全二进制文件中。在这种情况下，步骤 2.、3. 和 5. 仅适用于安全项目的配置。其他步骤会同时影响安全和非安全项目。此外，如果从非安全状态执行 STL，则用户必须实现安全网关特性，才能访问并调用在安全部分嵌入的所有 API。*

图 8. IAR™ 构建后操作屏幕截图

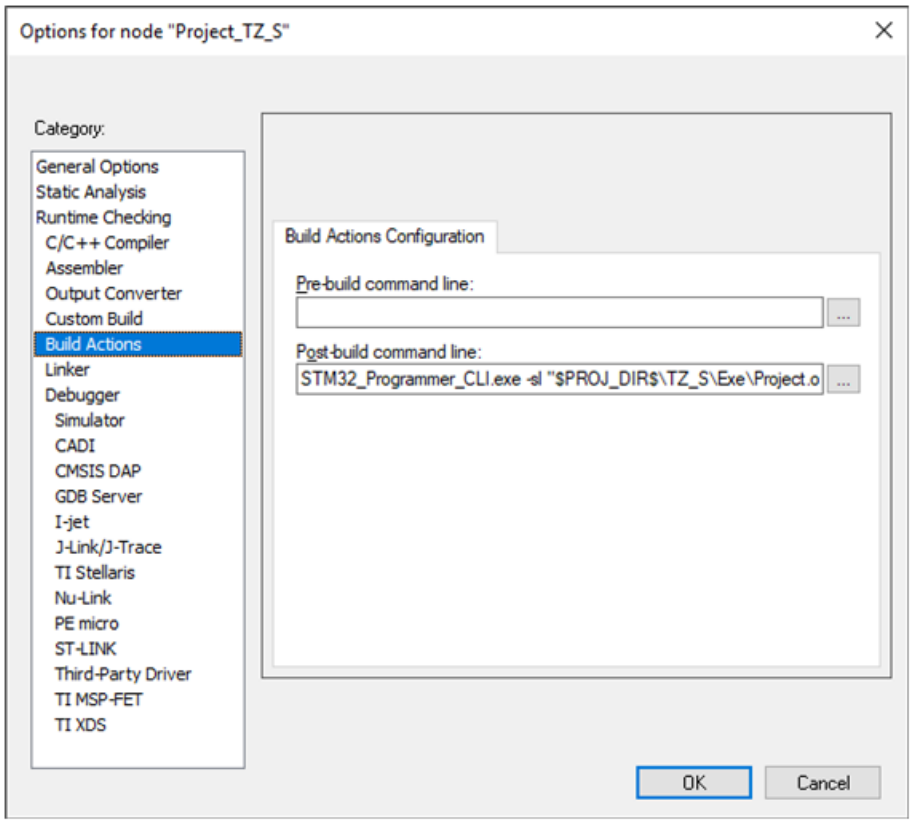
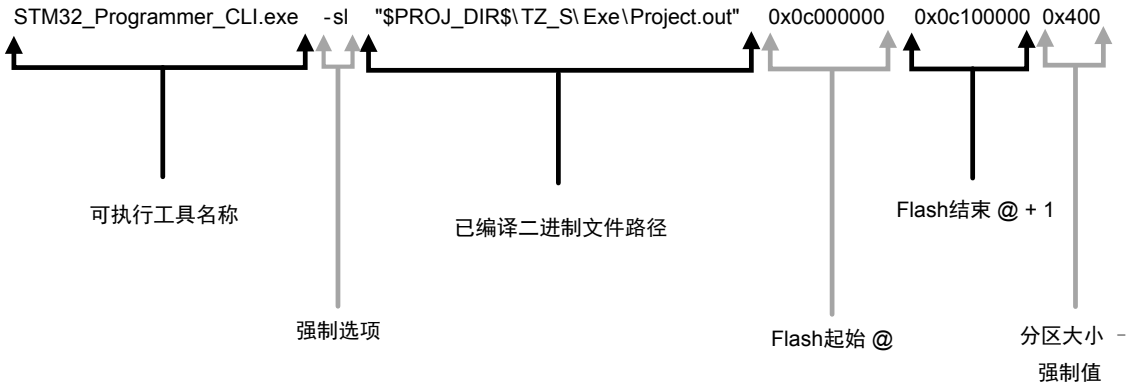


图 9. *CRC* 工具命令行



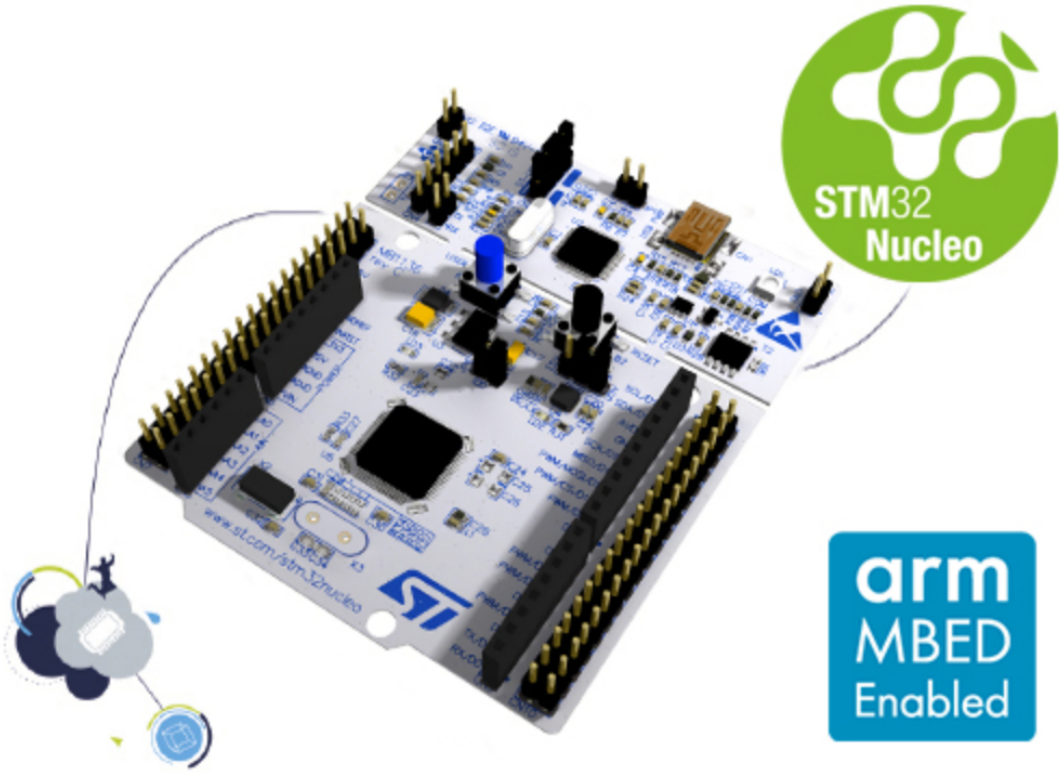
6 硬件和软件环境设置

6.1 硬件设置

STM32 Nucleo 板提供了平价且灵活的方案，用户可以使用任何 STM32 微控制器系列尝试新理念并构建原型。ARDUINO®连接支持和 ST Morpho 头使用户能够轻松地使用各种专用扩展板扩展 STM32 Nucleo 开放式开发平台的功能。由于集成了 ST-LINK/V2-1 调试器/编程器，STM32 Nucleo 板无需单独的探头。STM32 Nucleo 板附带 STM32 综合软件 HAL 库和各种套装软件示例。

关于 STM32 Nucleo 板的详细信息可从 http://www.st.com/stm32nucleo 网页上获取。

图 10. STM32 Nucleo 板示例



需要下列组件：

- STM32 NUCLEO-U575ZI-Q 开发板
- USB Type A 转 Micro-B USB 连接线，用于连接 STM32 Nucleo 和 PC。

6.2 软件设置

本节为开发人员列出了设置 SDK、运行示例场景以及自定义应用的最低要求。

6.2.1 开发工具链和编译器

选择一个 STM32Cube 软件扩展包支持的 *IDE*。

阅读所选 *IDE* 供应商提供的系统要求和设置信息。

检查发布包中的项目 Release_Notes.html 文件，并参考 *IDE* 兼容性一章（如果有）。

6.2.2 *CRC* 工具设置

意法半导体以 STM32CubeProgrammer 内部一项功能的形式提供 *CRC* 工具，用于 Flash 存储器测试。可使用其他 *CRC* 工具，前提是这些工具满足预期 CRC 的预计算一节中描述的要求。

工具安装步骤：

1. 在网站 www.st.com 提供的专用网页上选择 STM32CubeProgrammer
2. 安装软件包。

最简单的方式是在环境变量中添加工具路径（需要计算机管理权限）。如果没有，则必须在项目的构建后选项中直接添加路径，以便进行编译。

7 STL：用户 API 和状态机

7.1 用户结构

结构在文件 stl_user_api.h 中定义。禁止修改该文件的内容。
下面描述的结构是 stl_user_api.h 内容的副本：

```
typedef enum
{
  STL_OK = STL_OK_DEF, /* 调度器函数执行成功 */
  STL_KO = STL_KO_DEF /* 调度器函数执行未成功
                         防御性编程错误、校验和错误）。在这种情况下，
                          STL_TmStatus_t 值无关紧要 */
} STL_Status_t; /* STL 函数执行的状态返回值的类型 */
```

```
typedef enum
{
  STL_PASSED = STL_PASSED_DEF,  /* 测试通过。对于 Flash/RAM，测试通过并到达
                                   配置的末尾 */
  STL_PARTIAL_PASSED = STL_PARTIAL_PASSED_DEF, /* 只用于 RAM 和 Flash 测试。
                                                  测试通过，但尚未到达 Flash/RAM
                                                  配置的末尾
  STL_FAILED = STL_FAILED_DEF, /* 测试模块检测到硬件错误 */
  STL_NOT_TESTED = STL_NOT_TESTED_DEF, /* 软件初始化、软件配置、
                                          软件复位、软件去初始化后的初始值或
                                          测试模块未执行时的值。
  STL_ERROR = STL_ERROR_DEF /* 测试模块执行未成功（防御性编程
                               检查失败） */
} STL_TmStatus_t; /* 测试模块的结果的类型 */
```

```
typedef enum
{
  STL_CPU_TM1L_IDX = 0U,      /* CPU Arm 内核测试模块 1L 索引 */
  STL_CPU_TM7_IDX,       /* CPU Arm 内核测试模块 7 索引 */
  STL_CPU_TMCB_IDX,       /* CPU Arm 内核测试模块 B 类索引 */
  STL_CPU_TM_MAX         /* CPU Arm 内核测试模块的数量 */
} STL_CpuTmxIndex_t;     /* CPU Arm 内核测试模块的索引的类型
                            模块 */
```

```
typedef enum
{
  STL_CRC_TABLE_NO_TZ = STL_CRC_TABLE_NO_TZ_DEF,    /* Trust Zone 禁用时的 CRC 表索引 */
  STL_CRC_TABLE_TZ_S = STL_CRC_TABLE_TZ_S_DEF,      /* Trust Zone 启用且子集位于安全二进制文件中时的
CRC 表索引 */
  STL_CRC_TABLE_TZ_NS = STL_CRC_TABLE_TZ_NS_DEF     /* Trust Zone 启用且子集位于非安全二进制文件中时
的 CRC 表索引 */
} STL_CrcTableIdx_t;
```

```
typedef struct STL_MemSubset_struct
{
  uint32_t StartAddr; /* Flash 或 RAM 存储器子集的起始地址 */
  uint32_t EndAddr; /* Flash 或 RAM 存储器子集的结束地址 */
  STL_CrcTableIdx_t CrcTableIdx; /* CRC 表的索引（由 Flash TM 使用）*/
  struct STL_MemSubset_struct *pNext; /* 指向下一个 Flash 或 RAM 存储器子集的指针
                                         - 对于最后一个子集，必须设置为 NULL。*/
} STL_MemSubset_t; /* 用于定义要测试的 Flash
                      或 RAM 子集的类型 */
```

```
typedef struct
{
  STL_MemSubset_t *pSubset; /* 指向要测试的 Flash 或 RAM 存储器子集的指针 */
  uint32_t NumSectionsAtomic; /* 自动测试期间要测试的 Flash 或 RAM 分区
                                 的数量 */
} STL_MemConfig_t; /* 用于完整定义 Flash 或 RAM 测试配置的类型 */
```

```
typedef struct
{
  STL_TmStatus_t aCpuTmStatus[STL_CPU_TM_MAX]; /* CPU 测试模块的
                                                  强制状态值数组 */
  STL_TmStatus_t FlashTmStatus; /* Flash 测试模块的强制状态值 */
  STL_TmStatus_t RamTmStatus; /* RAM 测试模块的强制状态值 */
} STL_ArtifFailingConfig_t; /* 用于将每个 STL 测试模块的测试模块状态强制为
                               特定值的类型 */
```

提示 *请参考最新结构的 `stl_user_api.h` 文件（可从 www.st.com 获取）。*

7.2 用户 API

以下 API 在文件 ***stl_user_api.h*** 中定义。禁止修改该文件的内容。

Caution: 对于由用户应用程序定义并用作 ***STL API*** 参数的指针，用户应用程序必须设置有效指针，维持指针可用性，并检查指针完整性。***STL*** 不复制指针内容，并直接访问由应用程序定义的存储器地址。

这适用于整个 ***STL*** 执行过程。例如，对于保存了存储器测试配置的结构内容，必须保留用于访问该结构内容的指针。在调用与这些测试相关的 ***API*** 时，虽然它们不一定是输入参数列表的一部分，但仍然会被 STL_SCH_run_xxx 函数使用。

关于合适的 ***API*** 顺序调用的详细信息，请参见第 7.3 节 状态机 和第 7.6 节 测试示例。

7.2.1 常用 API

下面几节提供了常用 *API* 的详细信息。

7.2.1.1 *STL_SCH_Init*

描述：初始化调度器。可随时使用它将调度器去初始化（它复位所有测试）。

声明：`STL_Status_t STL_SCH_Init(void)`。

表 6. STL_SCH_Init 输入信息

允许的状态	参数
CPU TMx：全部 Flash *TM*：全部 *RAM TM*：全部	-

表 7. STL_SCH_Init 输出信息

`STL_Status_t` 返回值		返回的状态
值	注释	
STL_OK	函数执行成功	CPU TMx：CPU_TMx_CONFIGURED Flash *TM*：FLASH_IDLE *RAM TM*：RAM_IDLE
STL_KO	防御性编程错误的来源： • *STL* 内部数据损坏	无状态变化

附加信息：*CPU* 测试模块没有特定的 *CPU* 初始化函数。

提示 *该函数按照第 4.3.3 节 CRC 资源中的描述使用硬件 CRC。*

7.2.2 CPU Arm®内核测试 API

7.2.2.1 *STL_SCH_RunCpuTMx*

描述：运行 *CPU* 测试模块中的一个。

声明：`STL_Status_t STL_SCH_RunCpuTMx(STL_TmStatus_t *pSingleTmStatus)`，其中的 `TMx` 可以是 TM1L、TM7 和 TMCB 中的一个。

表 8. STL_SCH_RunCpuTMx 输入信息

允许的状态	参数	
	值	注释
CPU_TMx_CONFIGURED	`*pSingleTmStatus`	参见注意

表 9. **STL_SCH_RunCpuTMx 输出信息**

STL_Status_t 返回值		*pSingleTmStatus 输出		返回的状态
值	注释	值	注释	
STL_OK	函数执行成功	STL_PASSED	-	CPU_TMx_CONFIGURED
		STL_FAILED	-	
		STL_ERROR	防御性编程错误的来源： • *STL* 内部数据损坏 • 对 *CPU* TM7 未以特权级执行软件	
STL_KO	防御性编程错误的可能来源： • pSingleTmStatus = NULL • *STL* 内部数据损坏	无关	不得使用值	无状态变化

7.2.3 Flash 存储器测试 API

7.2.3.1 *STL_SCH_InitFlash*

描述：初始化 Flash 存储器测试。

声明：`STL_Status_t STL_SCH_InitFlash(STL_TmStatus_t *pSingleTmStatus)`

表 10. **STL_SCH_InitFlash 输入信息**

允许的状态	参数	
	值	注释
FLASH_IDLE FLASH_INIT FLASH_CONFIGURED	*pSingleTmStatus	注意

表 11. **STL_SCH_InitFlash 输出信息**

STL_Status_t 返回值		*pSingleTmStatus 输出		返回的状态
值	注释	值	注释	
STL_OK	函数执行成功	STL_NOT_TESTED	-	FLASH_INIT
STL_KO	防御性编程错误的可能来源： • pSingleTmStatus = NULL • *STL* 内部数据损坏	无关	不得使用值	无状态变化

7.2.3.2 *STL_SCH_ConfigureFlash*

描述：配置 Flash 存储器测试。

声明：`STL_Status_t STL_SCH_ConfigureFlash(STL_TmStatus_t *pSingleTmStatus, STL_MemConfig_t *pFlashConfig)`

表 12. **STL_SCH_ConfigureFlash 输入信息**

<table>
<tr><th rowspan="2">允许的状态</th><th colspan="4">参数</th></tr>
<tr><th>值</th><th colspan="3">注释</th></tr>
<tr><td rowspan="8">FLASH_INIT</td><td>*pSingleTmStatus</td><td colspan="3">参见注意</td></tr>
<tr><td rowspan="7">*pFlashConfig</td><td colspan="3">指向 Flash 存储器配置的指针。参见注意。</td></tr>
<tr><td>位域</td><td colspan="2">注释</td></tr>
<tr><td rowspan="4">*pSubset</td><td colspan="2">• 指向 Flash 存储器子集的指针。参见注意
• 分区不能与 CRC 区重叠</td></tr>
<tr><td>位域</td><td>注释</td></tr>
<tr><td>StartAddr</td><td>• 起始子集地址（字节）
• 不能低于 ROM_START 且不能高于 CRC_START 地址</td></tr>
<tr><td>EndAddr</td><td>• 结束子集地址（字节）
• 不能低于 ROM_START 且不能高于 CRC_START 地址
• 需高于 StartAddr</td></tr>
<tr><td>*pNext</td><td>• 指向下一个 Flash 存储器子集的指针。参见注意
• 对于最后一个子集，必须设置为 NULL</td></tr>
<tr><td>NumSectionsAtomic</td><td colspan="2">• 自动测试期间要测试的 Flash 存储器分区的数量
• 置为 1，即最小值（每次测试一个分区）
• 如果值大于所有子集中的分区数，则一次性测试所有 Flash 存储器子集</td></tr>
</table>

表 13. **STL_SCH_ConfigureFlash 输出信息**

STL_Status_t 返回值		*pSingleTmStatus 输出		返回的状态
值	注释	值	注释	
STL_OK	函数执行成功	STL_NOT_TESTED	-	FLASH_CONFIGURED
		STL_ERROR	防御性编程错误的可能来源： • 不允许的状态 • 检测到错误配置	无状态变化
STL_KO	防御性编程错误的可能来源： • pSingleTmStatus = NULL • pFlashConfig = NULL • STL 内部数据损坏	无关	不得使用值	无状态变化

附加信息：如果返回值置为 STL_KO 或*pSingleTmStatus 置为 STL_ERROR，则不应用 Flash 存储器配置。

7.2.3.3 *STL_SCH_RunFlashTM*

描述：运行 Flash 存储器测试。

声明：`STL_Status_t STL_SCH_RunFlashTM(STL_TmStatus_t *pSingleTmStatus)`

表 14. **STL_SCH_RunFlashTM 输入信息**

允许的状态	参数	
	值	注释
FLASH_CONFIGURED	`*pSingleTmStatus`	参见注意

表 15. **STL_SCH_RunFlashTM 输出信息**

STL_Status_t 返回值		`*pSingleTmStatus` 输出		返回的状态
值	注释	值	注释	
STL_OK	函数执行成功	STL_PASSED	-	FLASH_CONFIGURED
		STL_PARTIAL_PASSED	-	FLASH_CONFIGURED
		STL_FAILED	-	FLASH_CONFIGURED
		STL_NOT_TESTED	已测试所有子集	FLASH_CONFIGURED
		STL_ERROR	防御性编程错误的可能来源： • 不允许的状态 • 配置损坏	无状态变化
STL_KO	防御性编程错误的可能来源： • `pSingleTmStatus` = NULL • *STL* 内部数据损坏	无关	不得使用值	无状态变化

7.2.3.4 *STL_SCH_ResetFlash*

描述：复位 Flash 存储器测试。

声明：`STL_Status_t STL_SCH_ResetFlash(STL_TmStatus_t *pSingleTmStatus)`

表 16. **STL_SCH_ResetFlash 输入信息**

允许的状态	参数	
	值	注释
FLASH_CONFIGURED	`*pSingleTmStatus`	参见注意

表 17. **STL_SCH_ResetFlash 输出信息**

`STL_Status_t` 返回值		`*pSingleTmStatus` 输出		返回的状态
值	注释	值	注释	
STL_OK	函数执行成功	STL_NOT_TESTED	配置应用成功	FLASH_CONFIGURED
		STL_ERROR	防御性编程错误的可能来源： • 不允许的状态 • 配置损坏	无状态变化
STL_KO	防御性编程错误的可能来源： • `pSingleTmStatus` = NULL • *STL* 内部数据损坏	无关	不得使用值	无状态变化

附加信息

- 在测试了所有子集后，用户需将测试模块复位，以便再次执行 Flash 存储器测试。
- 如果返回值置为 STL_KO 或*pSingleTmStatus 置为 STL_ERROR，则不应用 Flash 存储器复位。

7.2.3.5 *STL_SCH_DeInitFlash*

描述：将 Flash 存储器测试去初始化 - 仅用于单一测试。

声明：`STL_Status_t STL_SCH_DeInitFlash(STL_TmStatus_t *pSingleTmStatus)`

表 18. STL_SCH_DeInitFlash 输入信息

允许的状态	参数	
	值	注释
FLASH_IDLE FLASH_INIT FLASH_CONFIGURED	*pSingleTmStatus	参见注意

表 19. STL_SCH_DeInitFlash 输出信息

STL_Status_t 返回值		*pSingleTmStatus 输出		返回的状态
值	注释	值	注释	
STL_OK	函数执行成功	STL_NOT_TESTED	-	FLASH_IDLE
STL_KO	防御性编程错误的可能来源： • pSingleTmStatus = NULL • STL 内部数据损坏	无关	不得使用值	无状态变化

7.2.4 RAM 测试 API

7.2.4.1 *STL_SCH_InitRam*

描述：初始化 *RAM* 测试。

声明：`STL_Status_t STL_SCH_InitRam(STL_TmStatus_t *pSingleTmStatus)`。

表 20. STL_SCH_InitRam 输入信息

允许的状态	参数	
	值	注释
RAM_IDLE RAM_INIT RAM_CONFIGURED	*pSingleTmStatus	参见注意

表 21. STL_SCH_InitRam 输出信息

STL_Status_t 返回值		*pSingleTmStatus 输出		返回的状态
值	注释	值	注释	
STL_OK	函数执行成功	STL_NOT_TESTED	-	RAM_INIT
STL_KO	防御性编程错误的可能来源： • pSingleTmStatus = NULL • *STL* 内部数据损坏	无关	不得使用值	无状态变化

7.2.4.2 *STL_Status_t STL_SCH_ConfigureRam*

描述：配置 *RAM* 测试。

声明：`STL_Status_t STL_SCH_ConfigureRam(STL_TmStatus_t *pSingleTmStatus, STL_MemConfig_t *pRamConfig)`

表 22. STL_SCH_ConfigureRam 输入信息

<table>
<tr><th rowspan="2">允许的状态</th><th colspan="4">参数</th></tr>
<tr><th>值</th><th colspan="3">注释</th></tr>
<tr><td rowspan="9">RAM_INIT</td><td>*pSingleTmStatus</td><td colspan="3">参见注意</td></tr>
<tr><td rowspan="8">*pRamConfig</td><td colspan="3">该指针包含 RAM 配置。参见注意</td></tr>
<tr><td>位域</td><td colspan="2">注释</td></tr>
<tr><td rowspan="4">*pSubset</td><td colspan="2">• 指向 RAM 子集的指针。参见注意
• 子集不能与 RAM 备份缓冲区重叠</td></tr>
<tr><td>位域</td><td>注释</td></tr>
<tr><td>StartAddr</td><td>• 起始子集地址（字节）
• 起始地址必须是 32 位对齐
• RAM 子集必须位于 RAM 区中
• 不能低于 RAM_START 且不能高于 RAM_END 地址</td></tr>
<tr><td>EndAddr</td><td>• 结束子集地址（字节）
• 高于 StartAddr
• 不能低于 RAM_START 且不能高于 RAM_END 地址
• 子集大小（EndAddr–StartAddr）需是 2 * RAM_BLOCK_SIZE（即 32 字节）的倍数
• 子集不能与 RAM 备份缓冲区重叠</td></tr>
<tr><td>*pNext</td><td>• 指向下一个 RAM 子集的指针。参见注意
• 对于最后一个子集，必须设置为 NULL</td></tr>
<tr><td>NumSectionsAtomic</td><td colspan="2">• 自动测试期间要测试的 RAM 分区的数量
• 置为 1，即最小值（每次测试一个分区）
• 如果值大于所有子集中的分区数，则一次性测试所有 RAM 子集</td></tr>
</table>

表 23. **STL_SCH_ConfigureRam 输出信息**

STL_Status_t 返回值		*pSingleTmStatus 输出		返回的状态
值	注释	值	注释	
STL_OK	函数执行成功	STL_NOT_TESTED	-	RAM_CONFIGURED
		STL_ERROR	防御性编程错误的可能来源： • 不允许的状态 • 检测到错误配置	无状态变化
STL_KO	防御性编程错误的可能来源： • pSingleTmStatus = NULL • pRamConfig = NULL • *STL* 内部数据损坏	无关	不得使用值	无状态变化

附加信息：如果返回值置为 STL_KO 或*pSingleTmStatus 置为 STL_ERROR，则不应用 *RAM* 配置。

7.2.4.3 *STL_SCH_RunRamTM*

描述：运行 *RAM* 测试。

声明：`STL_Status_t STL_SCH_RunRamTM(STL_TmStatus_t *pSingleTmStatus)`

表 24. **STL_SCH_RunRamTM 输入信息**

允许的状态	参数	
	值	注释
RAM_CONFIGURED	*pSingleTmStatus	参见注意

表 25. **STL_SCH_RunRamTM 输出信息**

STL_Status_t 返回值		*pSingleTmStatus 输出		返回的状态
值	注释	值	注释	
STL_OK	函数执行成功	STL_PASSED	-	RAM_CONFIGURED
		STL_PARTIAL_PASSED	-	RAM_CONFIGURED
		STL_FAILED	-	RAM_CONFIGURED
		STL_NOT_TESTED	已测试所有子集	RAM_CONFIGURED
		STL_ERROR	防御性编程错误的可能来源： • 不允许的状态 • 配置损坏	无状态变化
STL_KO	防御性编程错误的可能来源： • pSingleTmStatus = NULL • *STL* 内部数据损坏	无关	不得使用值	无状态变化

7.2.4.4 *STL_Status_t STL_SCH_ResetRam*

描述：复位 *RAM* 测试。

声明：`STL_Status_t STL_SCH_ResetRam(STL_TmStatus_t *pSingleTmStatus)`

表 26. **STL_SCH_ResetRam 输入信息**

允许的状态	参数	
	值	注释
RAM_CONFIGURED	`*pSingleTmStatus`	参见注意

表 27. **STL_SCH_ResetRam 输出信息**

`STL_Status_t` 返回值		`*pSingleTmStatus` 输出		返回的状态
值	注释	值	注释	
STL_OK	函数执行成功	STL_NOT_TESTED	配置应用成功	RAM_CONFIGURED
		STL_ERROR	防御性编程错误的可能来源： • 不允许的状态 • 配置损坏	无状态变化
STL_KO	防御性编程错误的可能来源： • `pSingleTmStatus` = NULL • *STL* 内部数据损坏	无关	不得使用值	无状态变化

附加信息

- 在测试了所有子集后，用户需要将测试模块复位，以便再次执行 *RAM* 测试。
- 如果返回值置为 STL_KO 或`*pSingleTmStatus` 置为 STL_ERROR，则不应用 *RAM* 复位。

7.2.4.5 *STL_SCH_DeInitRam*

描述：将 *RAM* 测试去初始化。

声明：`STL_Status_t STL_SCH_DeInitRam(STL_TmStatus_t *pSingleTmStatus)`

表 28. STL_SCH_DeInitRam 输入信息

允许的状态	参数	
	值	注释
RAM_IDLE RAM_INIT RAM_CONFIGURED	`*pSingleTmStatus`	参见注意

表 29. STL_SCH_DeInitRam 输出信息

`STL_Status_t` 返回值		`*pSingleTmStatus` 输出		返回的状态
值	注释	值	注释	
STL_OK	函数执行成功	STL_NOT_TESTED	-	RAM_IDLE
STL_KO	防御性编程错误的可能来源： • `pSingleTmStatus` = NULL • *STL* 内部数据损坏	无关	不得使用值	无状态变化

7.2.5 人为故障 API

7.2.5.1 *STL_SCH_StartArtifFailing*

描述：设置人为故障配置和启动人为故障功能。

声明：`STL_Status_t STL_SCH_StartArtifFailing(const STL_ArtifFailingConfig_t *pArtifFailingConfig)`

表 30. **STL_SCH_StartArtifFailing 输入信息**

允许的状态	参数	
	值	注释
CPU TMx： • CPU_TMx_CONFIGURED Flash *TM*： • FLASH_IDLE • FLASH_INIT • FLASH_CONFIGURED *RAM TM* • RAM_IDLE • RAM_INIT • RAM_CONFIGURED	`*pArtifFailingConfig`	-

表 31. **STL_SCH_StartArtifFailing 输出信息**

`STL_Status_t` 返回值	注释	输出	注释
STL_OK	函数执行成功	无输出参数	无状态变化
STL_KO	防御性编程错误的可能来源： • `pArtifFailingConfig` = NULL • 没有为每个测试模块设置配置值 • *STL* 内部数据损坏		

附加信息：除非 *STL_Status_t* 返回值置为 STL_OK，否则正常执行后续所有 *API* 调用，测试模块状态（`*pSingleTmStatus`、`*pTmListStatus`）强制为配置值。

7.2.5.2 *STL_SCH_StopArtifFailing*

描述：停止人为故障特性。

声明：`STL_Status_t STL_SCH_StopArtifFailing(void)`

表 32. **STL_SCH_StopArtifFailing 输入信息**

允许的状态	参数	
	值	注释
CPU TMx： • CPU_TMx_CONFIGURED Flash *TM*： • FLASH_IDLE • FLASH_INIT • FLASH_CONFIGURED *RAM TM* • RAM_IDLE • RAM_INIT • RAM_CONFIGURED	-	-

表 33. **STL_SCH_StopArtifFailing 输出信息**

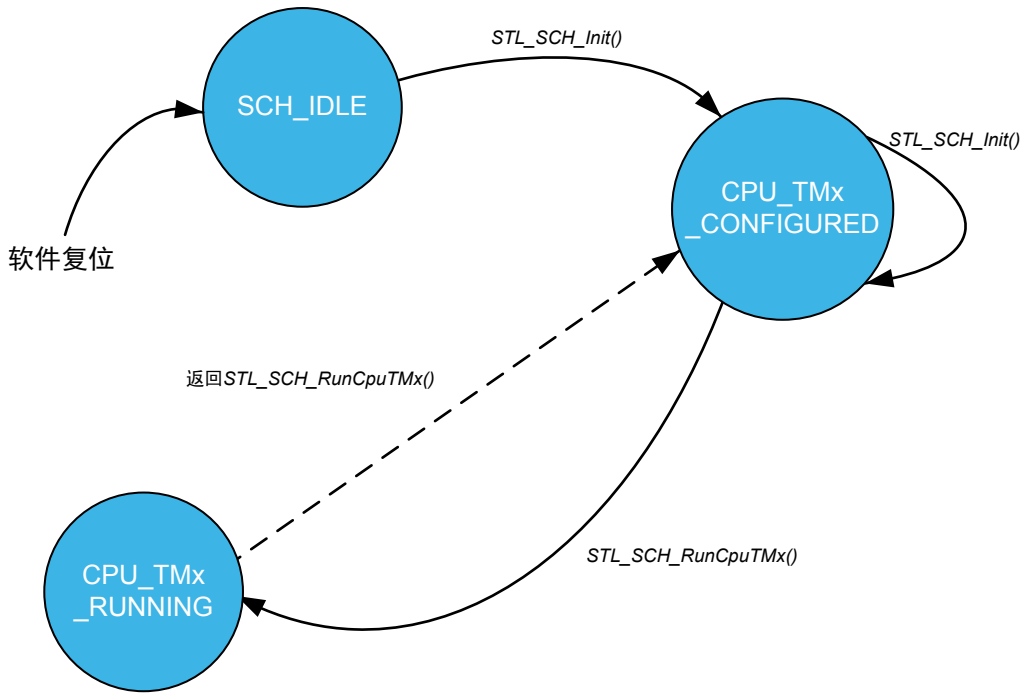
STL_Status_t 返回值	注释	输出	注释
STL_OK	函数执行成功	无输出参数	无状态变化
STL_KO	防御性编程错误的可能来源： • *STL* 内部数据损坏		

7.3 状态机

每个 *CPU* 测试模块都有自己的链接到 *CPU* 测试 *API* 的状态机图。

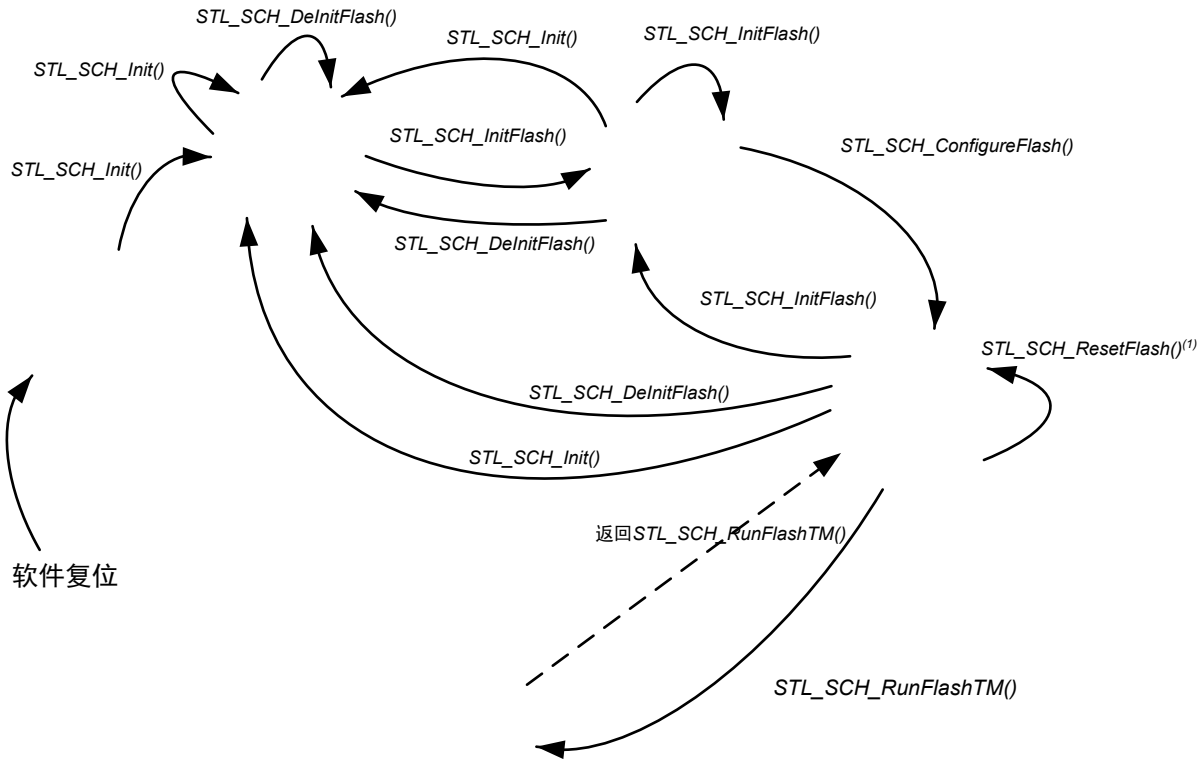
CPU* 测试 *API

图 11. **状态机图 - *CPU* 测试 *API***



Flash 存储器测试 *API*

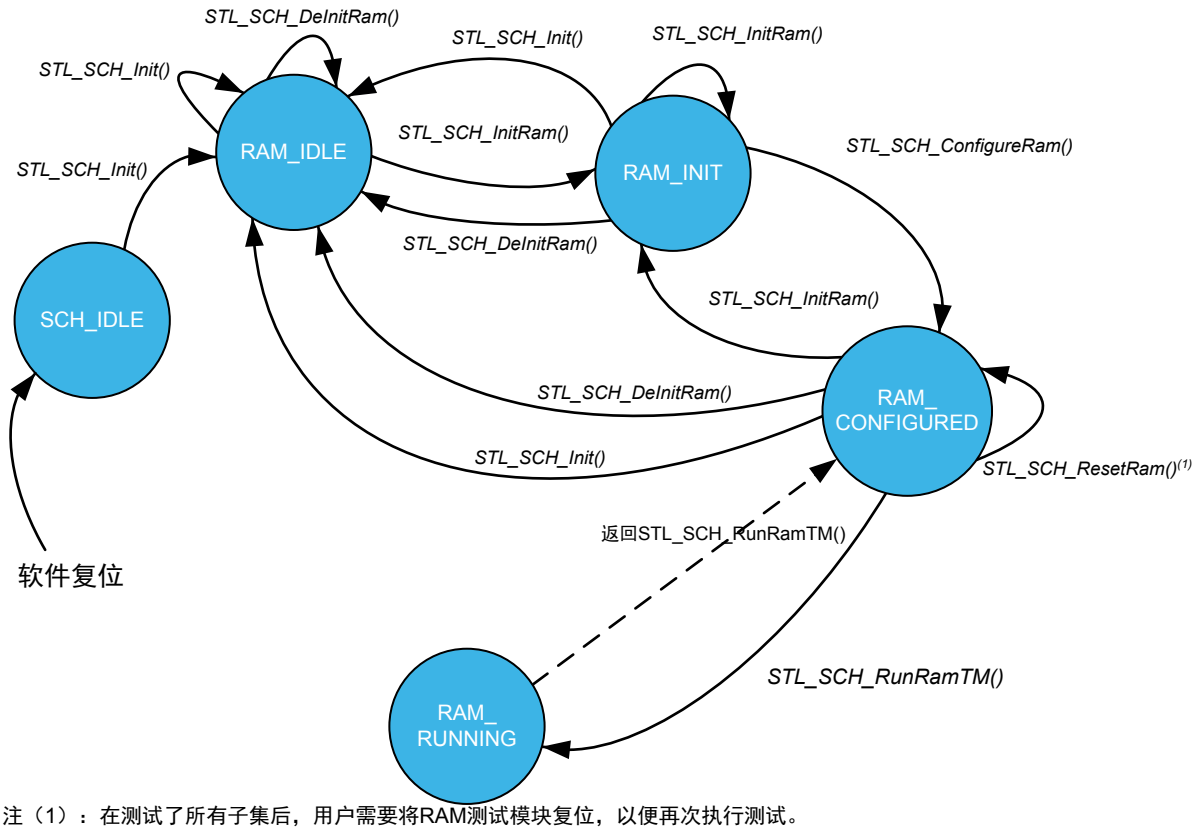
图 12. 状态机图 - Flash 存储器测试 *API*



注（1）：在测试了所有子集后，用户需要将Flash测试模块复位，以便再次执行测试。

RAM* 测试 *API

图 13. 状态机图 - *RAM* 测试 *API*



注（1）：在测试了所有子集后，用户需要将RAM测试模块复位，以便再次执行测试。

7.4 API 的使用和排序

用户应用程序必须：

- 维持在测试期间作为参数传递的指针的可用性和完整性。*STL* 不复制指针内容，并直接访问由应用程序定义的存储器地址。
- 在检查测试结果（`STL_TmStatus_t` 或 `STL_TmListStatus_t`）之前，检查函数返回值的状态（`STL_Status_t`）。参见交付的应用程序中的示例。

API 彼此*独立运行*，因此可按照任意顺序调用。

在执行任何存储器测试之前，*只有*这些测试的配置和初始化专用的 *API* 才是必须调用的。请参见第 7.3 节 状态机以获得更多信息。

测试流程经过简化，如今所有测试都从 C 代码执行，启动和运行时间测试之间再无任何间隔。该固件先前的版本中使用了此间隔。由于应用程序重启后缺少特定的初始测试，常规做法是在应用程序启动前执行完整的初始化序列，包括在所有存储区执行整套测试。此序列按以下顺序定义：

1. 所有 CPU 测试
2. 非易失性存储器的完整测试
3. 易失性存储器（包括栈的专用区）的完整测试。
4. 特定的客户测试

之后，在运行时，可用更宽松的方式修改和执行测试顺序。可以减少接受测试的存储器区域。甚至还可以在考虑其他因素（如可用的应用程序进程安全时间、系统总体性能、应用程序具体状态等）的同时，动态地修改测试过程。

7.5 用户参数

除了直接在 *API* 内部设置的参数，还有几个参数需要在 stl_user_param_template.c 文件中自定义。它们位于代码中，具有以下注释：

```
/* 可自定义 */
```

从 stl_user_param_template.c 提取：

```
/* Flash 配置 */
#define STL_ROM_NO_TZ_START_ADDR (0x08000000UL) /* 可自定义 - 无 TZ */
#define STL_ROM_TZ_S_START_ADDR (0x0C000000UL) /* 可自定义 - TZ，安全 Flash */
#define STL_ROM_TZ_NS_START_ADDR (0x08100000UL) /* 可自定义 - TZ，非安全 Flash */
#define STL_ROM_NO_TZ_END_ADDR (0x081FFFFFUL) /* 可自定义 - 无 TZ：2 Mbytes */
#define STL_ROM_TZ_S_END_ADDR (0x0C0FFFFFUL) /* 可自定义 - TZ，安全：1 Mbytes */
#define STL_ROM_TZ_NS_END_ADDR (0x081FFFFFUL) /* 可自定义 - TZ，非安全：1
Mbytes */
```

自定义特性取决于 STM32 产品和用户选择。

```
/* TM RAM 备份缓冲区配置 */
…
/* 用户应将缓冲区设置在 RAM 中 */
/* RAM 备份缓冲区位于“backup_buffer_section”中。*/
/* “backup_buffer_section”分区在分散加载描述文件中定义 */
```

自定义特性取决于用户选择。

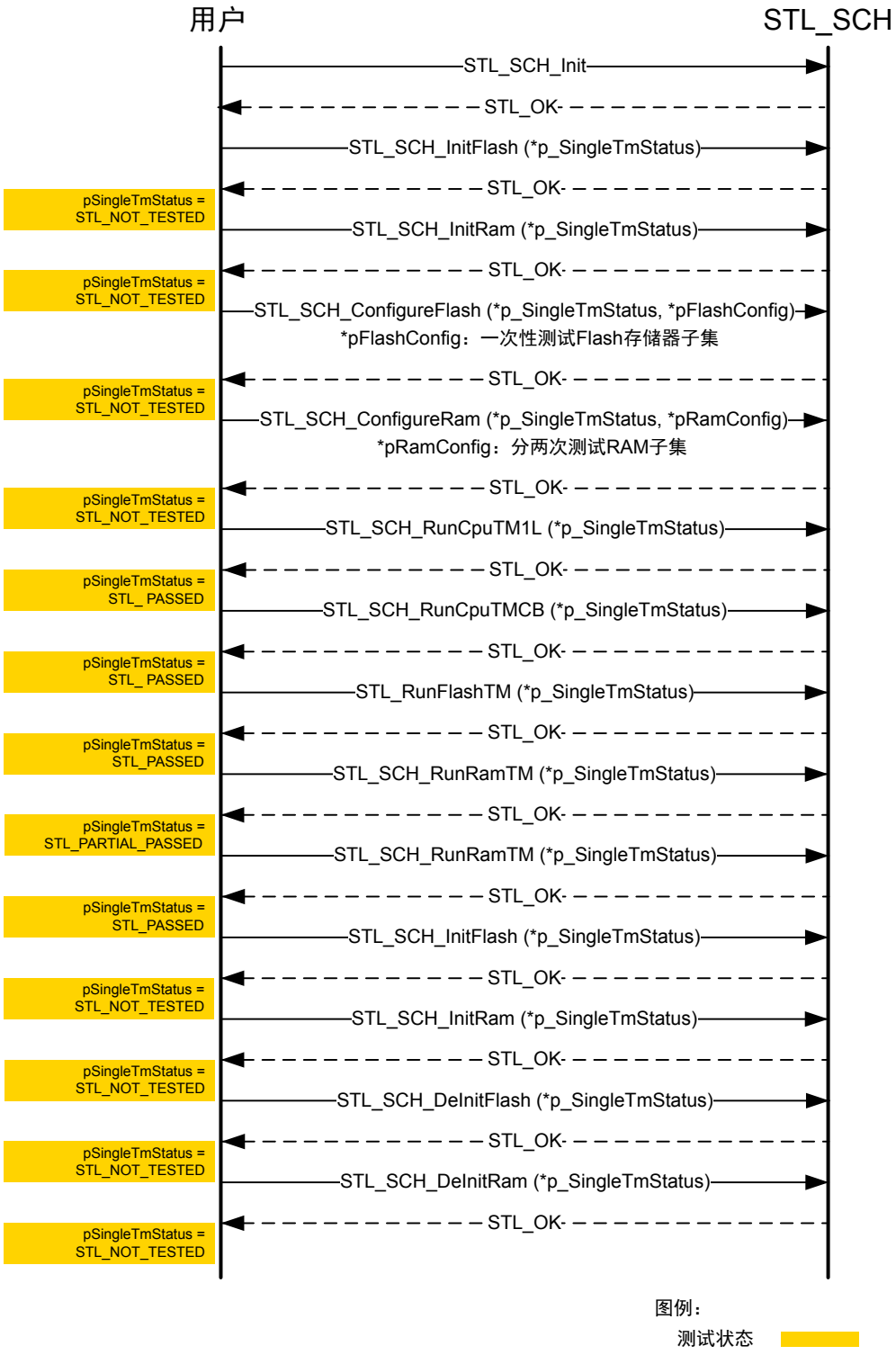
其余用户参数通过标记进行定义，可在以下文件中检查：

- stl_user_param_template.c：是否使用 *RAM* 备份缓冲区
- stl_util.c：使用软件还是硬件 *CRC* 计算
- stl_stm32_hw_config.h：若使用 *CRC* 硬件，则根据 STM32 器件选择正确的 *CRC* IP 配置

请参考第 5.5.2 节 从头开始构建应用程序的步骤了解标记配置检查。

7.6 测试示例

图 14. 单一测试示例



7.7 测试示例详情

7.7.1 Flash 存储器单一测试示例

图 15 所示为 Flash 存储器测试流程处理的示例：

- 使用两个 Flash 存储器子集
- 使用函数
  - STL_SCH_RunFlashTM → 只执行 Flash 存储器测试模块
  - STL_SCH_ResetFlash
- 函数返回值
- Flash 存储器测试模块结果值：`pSingleTmStatus` → 在这种情况下，它包含 Flash 存储器测试的结果

7.7.2 RAM 单一测试示例

图 16 所示为 *RAM* 测试流程处理的示例：

- 使用两个 *RAM* 子集
- 使用函数：
  - STL_SCH_RunRAMTM → 只执行 *RAM* 测试模块
  - STL_SCH_ResetRam
- 函数返回值
- *RAM* 测试模块结果值：`pSingleTmStatus` → 在这种情况下，它包含 *RAM* 存储器测试的结果

7.7.3 测试示例图

图 15. **Flash 存储器单一测试示例**

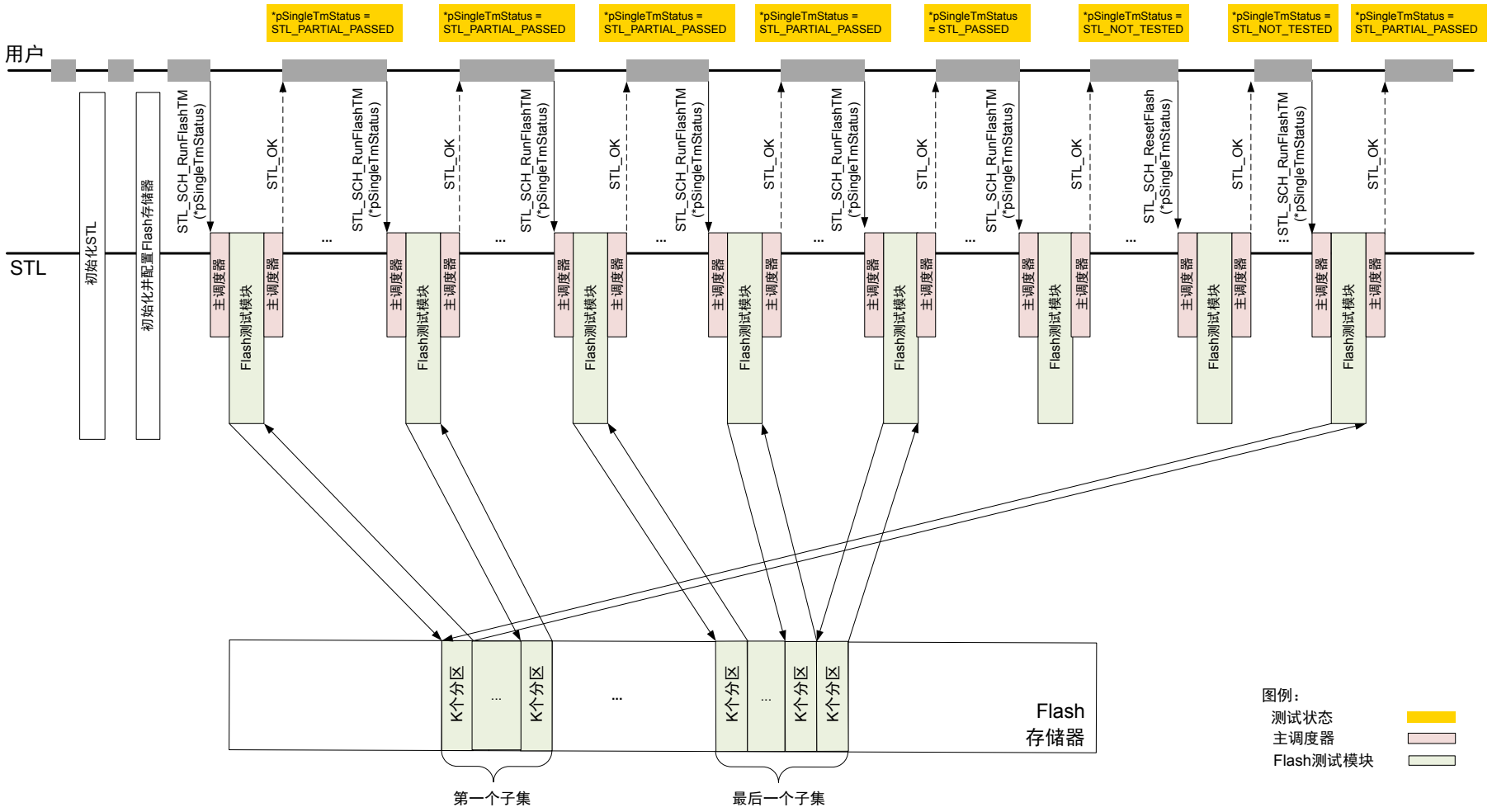
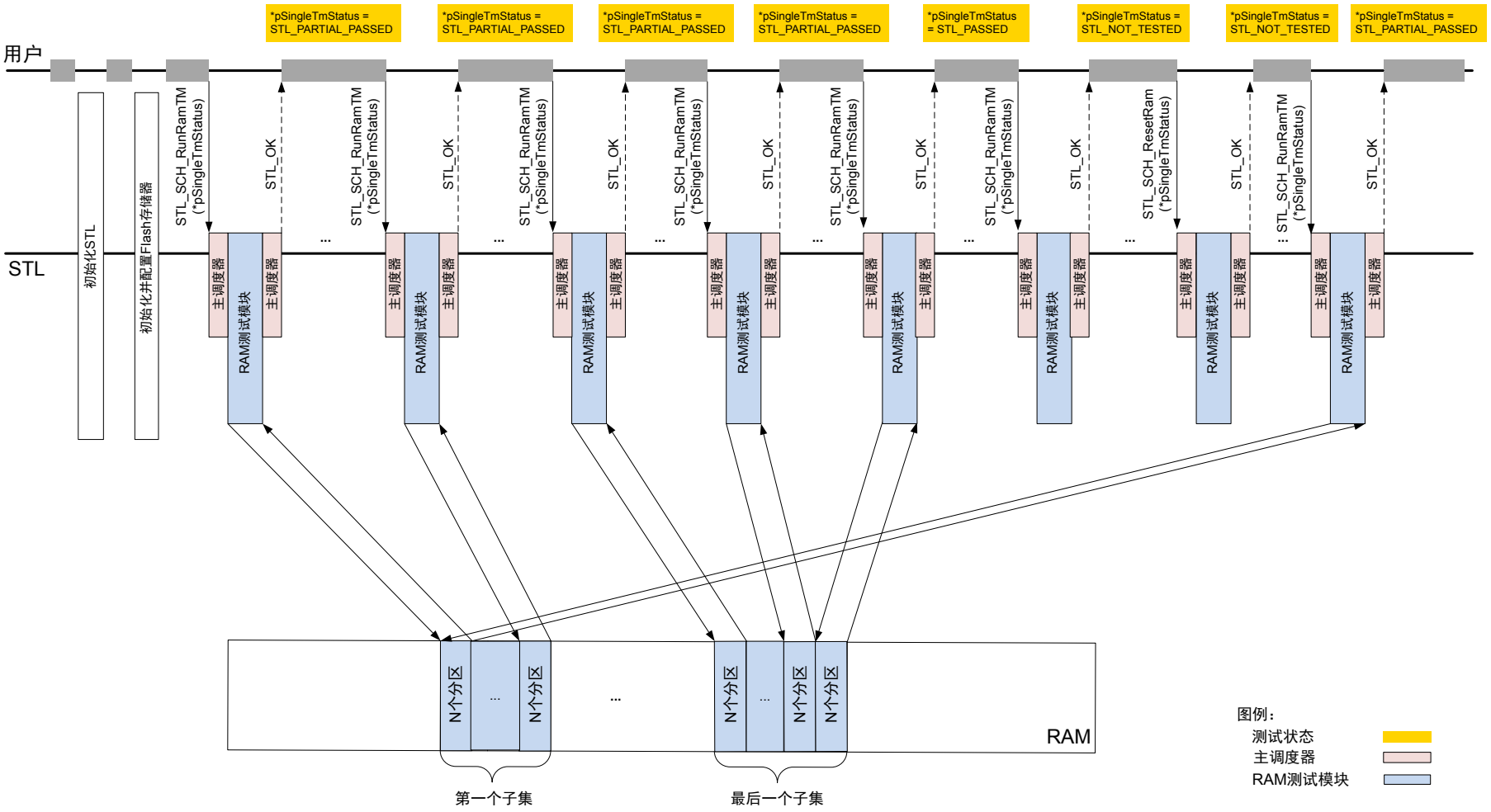


图 16. **RAM 单一测试示例**



8 STL：执行时间详情

表 34 中的数据是使用以下测试设置获得的：

- *STL* 第 5.5 节 应用程序：编译过程中提供了库的编译详情。
- 性能测试项目使用 Arm®（EWARM）工具链 v9.20.1 的 IAR 嵌入式工作台®进行编译
- 在以下条件下编译了软件配置：
  - HCLK 时钟设置为 160 MHz
  - Flash 存储器延迟设置为 4 个等待状态
  - *ICache* 激活
  - NUCLEO-U575ZI-Q 版本 C（MB1549 C）
  - *STL* 在安全二进制文件中运行，由在非安全二进制文件中运行的用户应用程序调用

表 34. 集成测试

测试	持续时间（单位：时钟周期）		测试的存储器
	硬件 CRC	软件 *CRC*(1)	
`STL_SCH_InitFlash()`	683	1119	-
`STL_SCH_ConfigureFlash()`	906	1325	-
`STL_SCH_RunFlashTM()`	51795	262587	• 在安全 Flash 存储器中测试了 11264 字节 • 在非安全 Flash 存储器中测试了 11264 字节
`STL_SCH_InitRam()`	846	1263	-
`STL_SCH_ConfigureRam()`	855	1277	-
`STL_SCH_RunRamTM()`(2)	27253686	27254029	在安全 *RAM* 中测试了 278528 字节并在非安全 RAM 中测试了 524288 字节 *RAM*
`STL_SCH_RunCpuTM1L()`	3326	3786	-
`STL_SCH_RunCpuTM7()`(2)	1664	2168	-
`STL_SCH_RunCpuTMCB()`	821	1277	-

1. *CRC 算法导致测试持续时间延长（参见第 4.1.1 节 调度器原理）。*
2. *默认配置：不使能 STL_ENABLE_IT。*

9 应用相关特定测试不在 ST 固件自检库范围内

用户必须注意其余所有必要的测试，包括不在 ST 固件库中的应用程序特定的 *MCU* 部件：

- 模拟部件（ADC/DAC 和复用器）的测试
- 数字 I/O 测试
- 外部寻址
- 外部通信
- 定时和中断
- 系统时钟频率测量。

提示 *时钟频率测量不属于 STL 软件包的集成部分。时钟测试模块在 STL 集成示例中以开源的方式提供，演示了实现用户定义的可包含在 STL 流程中的额外测试模块的能力。有关详细信息，请参见第 9.5 节 STL 库扩展能力。*

这些组件的有效解决方案在很大程度上取决于应用及外设的能力。从设计的最早阶段开始，应用程序必须尽可能精确地遵守推荐的测试原理。

该方法经常会导致硬件和软件层级的冗余。

硬件方法可基于：

- 输入和/或输出的倍增
- 参考点测量
- 模拟或数字输出（如 *DAC*、*PWM*、GPIO）上的环回读取控制 *GPIO*
- 配置保护。

软件方法可基于：

- 在不同时间执行或通过不同方法执行的多次重复、多次采样、多次检查、决策或计算
- 数据冗余（数据复制、奇偶校验、错误修正/检测代码、校验和、拟定协议）
- 真实性检查（有效范围、有效组合、预期变化或趋势）
- 周期性和发生率检查（流程和定时控制）
- 正确配置的定期检查（如读回配置寄存器）。

9.1 模拟信号

必须检查测量值的一致性，并通过在其他冗余通道上执行测量来进行验证。在测试应用程序中使用的模拟复用器时，可使用空闲通道读取某些参考电压。此外，还须检查内部参考电压。

一些 STM32 微控制器器件具有两个（甚至三个）独立的 *ADC* 模块。为确保结果的可靠性，出于安全原因，使用两个不同 *ADC* 模块在同一个通道上执行多次转换。使用以下方式之一获取结果：

- 从一个通道多次采样
- 比较冗余通道，然后进行平均运算。

下面是对在 STM32 微控制器器件上测试模拟部件功能的一些提示。

ADC 输入引脚断开

可通过在受试引脚上施加额外信号源来测试 ADC 输入引脚断开。

- 一些 STM32 微控制器器件的模拟输入上具有内部下拉或上拉电阻激活装置。它们还可能具有带 *DAC* 功能或数字 *GPIO* 输出的空闲引脚。这些引脚中的任何一个都可用作 ADC 的已知参考输入。
- 一些 STM32 微控制器器件具有路由接口。此接口可用于引脚之间的内部连接，以便执行：
  - 环回测试
  - 额外的信号注入
  - 在其他一些独立通道上重复测量。

提示 *用户必须防止任何危险电压注入模拟引脚。在组合数字和模拟信号时和对模拟和数字部件施加不同电平（$V_{DD} > V_{DDA}$）时，可能发生这种情况。*

内部参考电压和温度传感器（某些器件为 V_{BAT}）

- 可验证这些信号的比值是否在允许范围内。
- 如果 V_{DD} 电压已知，可执行额外测试。

ADC 时钟

ADC 转换时间的测量（通过定时器）可用于测试独立的 *ADC* 时钟功能。

DAC 输出功能

可使用空闲的 *ADC* 通道检查 *DAC* 输出通道是否正常工作。
在连接 *ADC* 输入通道和 *DAC* 输出通道时，可使用路由接口。

比较器功能

已知电压与 *DAC* 输出或内部参考电压之间的比较可用于测试另一个比较器输入上的比较器输出。
通过激活受试引脚上的下拉或上拉电阻并将此信号与另一个比较器输入上作为参考的 *DAC* 电压进行比较，可以测试模拟信号断开。

运算放大器

为了测试功能，可测量一个已知模拟信号或将其强制到运算放大器（*OPAMP*）输入引脚，并使用 *ADC* 从内部测量输出电压。*OPAMP* 的输入信号还可以通过 *ADC* 测量（在另一个通道上）。

9.2 数字 I/O

B 类测试必须检测数字 I/O 上的任何故障。它可能与其他一些应用程序部件一道接受真实性检查。例如，在开启/关闭加热/冷却数字控制功能时，必须检查温度传感器的模拟信号变化。通过对 GPIOx_LCKR 寄存器中的锁定位使用正确的锁定序列，可以锁定选定的端口位。此操作可防止端口配置被意外更改。在这种情况下，仅可在下一个复位序列进行重新配置。此外，位带特征可用于 SRAM 和外围寄存器的原子操纵。

9.3 中断

必须检查事件的发生率和周期性。可采用不同的方法；一种方法是，当每个中断事件使特定定时器递增时，采用一套增量计数器。通过其他独立的时基定期对计数器内的值进行交叉检查。上个周期内发生的事件数量取决于应用要求。
配置锁定功能可用于保护定时器寄存器设置，它具有由 TIMx_BDTR 寄存器控制的三个级别。未使用的中断矢量必须转移到常见错误处理程序中。如果能简化应用中断方案，对于非安全相关问题最好进行轮询。

9.4 通信

在将冗余信息纳入数据包时，必须检查通信会话期间的数据交换。为了这一目的，可使用奇偶校验、同步信号、CRC 校验和、块重复或协议编号。如果必要，稳定的应用软件协议栈（如 TCP/IP）可提供更高级的保护。必须一直检查通信事件的周期性和发生率及协议错误信号。
用户可在产品专用安全手册中找到更多信息和方法。

9.5 STL 库扩展能力

相比于该 *STL* 库先前的版本（参见第 1.2 节），此框架版本的实现要容易得多且更灵活，更容易进行扩展。即使采用了新形式，此框架也能确保相同自检方法集合（库的先前版本已经实现）符合 IEC 60730 标准：

- 在 *CPU TM* 测试寄存器
- Flash TM 的 32 位 *CRC* 计算兼容 STM32 硬件 *CRC* 单元 *TM*
- March C 测试遵循 RAM TM 的物理地址顺序 *RAM TM*
- LSI 触发的定时器检查 *TM* 时钟（在 *STL* 集成示例中定义）的系统时钟频率

新框架版本的主要改进：

- 以模块为导向
- 支持部分测试
- 基于配置和参数化结构
- 启动与运行时间测试之间无差异
- *CRC* 计算支持基于 STM32CubeProgrammer 命令行功能提供的格式
- 关键通用模块的预编译和固定对象代码格式
- 不依赖于驱动程序或编译器上的通用模块执行
- 人为故障控制功能，用于确认模块的正确集成，无需额外的插装代码
- 通过额外的应用程序特定的模块轻松完成扩展。

固件包集成示例中提供了额外的特定测试模块实现的示例。以两个独立时钟源的交叉检查测量法为基础的特定测试模块以开源的形式提供，附带固件包集成示例。终端用户必须修改此模块，将所用时钟系统的配置上的特定依赖关系计算在内。

此模块使用的测量原理与先前库版本中已使用的相同。用于频率比较的硬件最初必须被配置（TIM16 的通道 1 被 LSI 触发）为在调用相关 *API* 之前调用时钟测量。此硬件配置在 `main.c` 文件中 `STL_Init()` 程序结束时完成。*API* 被编写成使用与 *STL* 中集成的常规 *API* 兼容的接口，因此声明中使用相同格式：

```
STL_Status_t STL_SCH_RunClockTest(STL_TmStatus_t *pSingleTmStatus)
```

在该函数调用期间传递的参数充当指向时钟模块测量状态的指针，并且函数本身提供 STL_KO vs STL_OK 返回状态，如果发生防御性编程错误，常规 *STL* 模块同样如此。如果时钟测量硬件激活且最后一个测量周期（（置为 8 个连续 LSI 周期））更新的新周期值处于预期区间（通过宏 `CLK_LimitLow` 和 `CLK_LimitHigh` 定义）以内，则模块测量状态值变为 STL_PASSED。否则按照常规 *API* 模块置为 STL_FAILED。如果调用了模块的人为故障，也同样如此。

同样地，用户可以集成以下模块。例如，此新软件包默认不再包含栈指针有效性检查或看门狗测试和维护的实现。该库的更早版本提供了这些测试的源代码，参见第 1.2 节 。关于家居标准未特别要求的公认安全方法的更多信息，请参考第 1.2 节 。此类安全方法可能有助于改善用户应用程序的稳健性。

10 符合 IEC、UL 和 CSA 标准

核心 IEC 标准为 IEC 60730-1 和 IEC 60335-1，从第 4 版开始与 UL/CSA 60730-1 和 UL/CSA 60335-1 进行了协调。此外，之前的 UL/CSA 版本使用对 UL1998 标准的引用。

标准会定期更新。标准中收集的全部法规的范围极广；与微控制器通用部件的软件自检要求相关的章节是特定的。在大多数情况下，提供的更新完全不影响标准的这些特定部分。因此，过时的认证可能仍然符合并且对标准的更新版本有效。

以下附录中定义了需要满足的相关具体条件：

- IEC 60335-1 标准的附录 Q 和 R
- IEC 60730-1 标准的附录 H。

IEC 60730-1:2010 H.2.22 定义了 3 个类别，它们是：

- A 类：不依赖应用安全的控制功能。
- B 类：旨在防止受控设备的不安全状态的控制功能。控制功能故障不会直接导致危险情况。
- C 类：旨在防止特殊危险（例如爆炸或其故障可直接导致装置中发生危险）的控制功能。

对于应用了安全保护功能的可编程电子组件，IEC 60335-1 标准要求包含软件措施，以便控制表 R.1 和 R.2 中指定的故障和错误状态：

- 表 R.1 总结了可与 B 类的给定要求相比的一般条件
- 表 R.2 总结了可与 C 类的给定要求相比的特定条件。

对 B 类软件的要求是本用户手册的主题，定义这些要求的目的是在电器中其他位置发生另一个故障时预防危险。在这种情况下，在发生故障后，在电器上运行自检软件。由于使用了另一个冗余软件程序或该级别要求的硬件保护功能，在安全关键型例程执行期间发生意外的软件故障不一定会导致危险情况。

C 类不要求此类硬件保护，原因在于安全关键型软件中发生任何故障都可能导致潜在危险情况。为了符合该类别的要求，需要比通常适用于标准工业微控制器（如 STM32）的测试更稳健的测试。可接受的解决方案通常趋向于在系统层面实现特定硬件冗余，如双通道结构。

关于更严格测试方法的更多信息，请参考第 1.2 节 中的行业文档。

IEC 60730-1 定义了 B 类控制功能设计可接受的适用架构的集合：

- 单通道有功能测试。一个 CPU 按需要执行软件控制功能。在软件启动时执行功能测试。它保证所有关键功能正常工作。
- 单通道有定期自测。一个 CPU 执行软件控制功能。内置的定期测试检查系统的各种关键功能，同时不影响规划控制任务的性能。
- 带比较功能的双通道（同质或异质）。软件用于在两个独立 CPU 上执行控制功能（同质或异质）。在执行任何安全关键型任务时，两个 CPU 比较内部信号以便进行故障检测。

提示 *此结构也被公认为符合 C 类的要求。一个共同原则是，凡是符合 C 类的要求的方法都自动符合 B 类的要求。*

下表汇总了 *STL* 应用的方法及其对标准的引用。*STL* 专注于在所有应用程序中重复使用的微控制器通用组件。其他部分的测试由终端用户负责，其测试大多与特定应用程序相对应，可在系统设计的计划阶段高效完成。请参考第 9 节 应用相关特定测试不在 ST 固件自检库范围内获取关于如何处理这些应用程序特定的测试的更多信息。

表 35. 采用 IEC-60730-1 认可的方法的 X-CUBE-CLASSB 库覆盖的 IEC 60335-1 组件

表 R.1 中的组件（IEC 60335-1：附录 R）		B 类	对 IEC 60730-1：附录 H 的引用	故障/错误	在 X-CUBE-CLASSB 上应用的安全方法 X-CUBE-CLASSB	注释
1. CPU	1.1 CPU 寄存器	X	H.2.16.5 H.2.16.6 H.2.19.6	固定型故障	定期运行 *STL* TM1L、TM7 和 TMCB CPU 测试模块	*CPU* 寄存器的功能模式测试，状态寄存器和栈指针的功能测试
	1.2 指令解码和执行	N/A				B 类不要求
	1.3 程序计数器	X	H.2.18.10.2 H.2.18.10.4	固定型故障	N/A 终端用户责任	逻辑和时隙程序序列监测，看门狗实现
	1.4 寻址	N/A				B 类不要求

表 R.1 中的组件（IEC 60335-1：附录 R）		B 类	对 IEC 60730-1：附录 H 的引用	故障/错误	在 X-CUBE-CLASSB 上应用的安全方法 X-CUBE-CLASSB	注释
1. CPU	1.5 数据路径指令解码	N/A				B 类不要求
2. 中断处理和执行		X	H.2.18.10.4 H2.18.18	无中断或中断过于频繁	在与时钟交叉检查测量模块相关的中断处应用结果握手	应用程序中实现的其他中断由终端用户负责
3. 时钟		X	H.2.18.10.1 H.2.18.10.4	频率错误	定期运行时钟交叉检查模块。在固件集成示例中以开源的形式添加为用户特定的测试模块	在两个独立的时钟源（系统时钟和 LSI）之间执行时钟交叉检查测量
4. 存储器	4.1 不可变存储器	X	H.2.19.3.1 H.2.19.3.2 H.2.19.8.2	所有一位故障	定期执行 *STL* FlashTM 测试模块	由终端用户负责 ECC 使能
	4.2.变量内存	X	H.2.19.6 H.2.19.8.2	DC 故障	定期执行 *STL* RamTM 测试模块	由终端用户负责 ECC 使能
	4.3 寻址（与可变和不可变存储器有关）	X	H.2.19.8.2	固定型故障	-	通过执行使用的存储器测试模块间接地进行测试 由终端用户负责 ECC 使能
5. 内部数据路径	5.1 数据	X	H.2.19.8.2	固定型故障	-	
	5.2 寻址	X	H.2.19.8.2	地址错误	-	
6. 外部通信		X	-	-	N/A 终端用户责任	-
7. I/O 外设		X	-	-	N/A 终端用户责任	-
8. 监测器件和比较器		N/A				B 类不要求
9. 定制芯片		X	-	-	N/A	-

版本历史

表 36. 文档版本历史

日期	版本	变更
2022 年 8 月 31 日	1	初始版本。

用语集

ADC 模数转换器

AEABI Arm® 嵌入式应用程序二进制接口

API 应用编程接口

APSR CPU 状态寄存器

BSP 板级支持包

B 类
家用电器安全法规（UL/CSA/IEC 60730-1/60335-1）的中间级别

CMSIS 通用微控制器软件接口标准

CPU 中央处理器

CRC 循环冗余校验

DAC 数模转换器

GPIO 通用输入输出

HAL 硬件抽象层

ICache 指令缓存

IDE 集成开发环境

LL 底层

MCU 微控制器单元

MSP 主堆栈指针

OPAMP 运算放大器

PSP 进程栈指针

PWM 脉冲宽度调制

RAM 随机存取存储器

SDK 软件开发套件

SG 安全网关

STL 自检库

TM 测试模块

目录

表一览

图一览

重要通知 - 仔细阅读

STMicroelectronics International NV 及其子公司（“ST”）保留随时对意法半导体产品和/或本文档进行变更、更正、增强、修改和改进的权利，恕不另行通知。

本文档仅用于获取与意法半导体产品有关的基本信息。因此，您在此同意，仅为获取与意法半导体产品的基本信信息而使用本文档。您进一步承认并同意，本文档不得用于或涉及任何法院、仲裁、机构、委员会或其他法庭的任何法律或行政程序，也不得涉及任何形式的诉讼、诉因、上诉、索赔、指控、法律要求或纠纷。您进一步承认并同意，不得将本文档解释为承认、确认或证实（包括但不限于）意法半导体或其任何附属公司的责任、过失或职责，或本文档信息的准确性或有效性，也不得解释为与任何所谓的产品问题、故障或缺陷有关。意法半导体对本文档的准确性不作任何承诺，并明确声明，对本文档信息准确性作出的任何明示或暗示保证概不负责。因此，您同意，在任何情况下，对于您因信赖或使用本文档而产生的或与之相关的任何直接性、间接性、必然性、偶然性、后果性、惩戒性、惩罚性损害或其他损害（包括利润损失），意法半导体或其公司均不承担责任。

买方在订货之前应获取关于意法半导体产品的最新信息。意法半导体产品的销售依照订单确认时的相关意法半导体销售条款，包括但不限于其中的保修条款。

在此方面，注意：意法半导体产品并非专用于上述条款所述的某些特定应用场合或环境。

买方自行负责对意法半导体产品的选择和使用，意法半导体概不承担与应用协助或买方产品设计相关的任何责任。

提供的信息被视为准确可靠的信息。但是，意法半导体对使用该信息的后果或因使用而侵犯专利或其它第三方权利的后果不承担任何责任。意法半导体不对任何知识产权进行任何明示或默示的授权或许可。

转售的意法半导体产品如有不同于此处提供的信息的规定，将导致意法半导体针对该产品授予的任何保证失效。

ST 和 ST 标志是意法半导体的商标。关于意法半导体商标的其他信息，访问 www.st.com/trademarks。其他所有产品或服务名称是其各自所有者的财产。

本文档中的信息取代本文档所有早期版本中提供的信息。