

STM32F4系列 UL/CSA/IEC 60730-1/60335-1 自检库用户指南

介绍

本文件适用于包含 Arm® Cortex®-M4 核心的 STM32F4 系列微控制器的 X-CUBE-CLASSB 自检库集。订单代码 X-CUBE-CLASSB-F4。

安全在电子应用中起着至关重要的作用。对元件的安全要求水平正在稳步上升，并且电子设备制造商在其设计中包含许多新的技术解决方案。提高安全性的技术

它们持续演进，并定期被纳入安全标准的更新版本中。{v*}

现行的安全建议和要求规定在由不同机构发布的全球标准中。这些包括：国际电工委员会（IEC）、美国保险商实验室（UL）以及加拿大标准协会（CSA）等机构。

合规、验证和认证是认证机构的重点工作。这些包括：德国TÜV和VDE（主要在欧洲开展业务），以及美国UL和加拿大CSA（主要针对美国和加拿大市场）。

与安全要求相关的标准具有非常广泛的应用范围。这些安全标准涵盖许多领域，例如：分类、方法论、材料、机械、标签、硬件和软件测试。在此，目标仅仅是确保可编程电子组件的软件要求符合性，这些组件构成了安全标准中的一个特定部分。这些要求在标准发布新版本时极易发生变化。此外，面向通用微控制器部件测试的安全标准之间存在显著的相似性，例如 CPU 或存储器。

本文件中呈现的库是基于ST开发并应用于满足严格IEC 61508工业安全标准要求的测试模块的部分子集。这些模块被调整以符合针对家庭安全的IEC 60730标准。这就是为什么这个新库采用了与之前版本不同的交付格式。该格式源自当前作为黑盒预编译对象交付的工业安全库，该对象不包含源代码但具有明确的外部接口定义。这种不可变解决方案的优势在于其与编译工具链无关。它还独立于任何其他固件，如HAL、LL或CMSIS层。该解决方案可防止当使用旧版本库验证过的源代码文件被新版本编译器重新编译或与最新固件驱动程序结合时出现意外的编译结果。这通常是常见的做法。

表1. 适用产品

Part number	Order code
X-CUBE-CLASSB	X-CUBE-CLASSB-F4



1 一般信息 {v*}

1.1 目的和范围

本文档适用于专为嵌有 Arm® Cortex®-M4 的 STM32F4 系列微控制器设计的 X-CUBE-CLASSB 自检库集。X-CUBE-CLASSB-F4 扩展包提供与应用无关的软件，以符合 UL/CSA/IEC 60730-1 安全标准。UL/CSA/IEC 60730-1 安全标准旨在确保与家用设备及类似电子应用配合使用的自动电气控制的安全性。

这个软件库的主要目的是促进和加速：

- 用户软件开发
- 受相关要求和认证约束的应用认证流程



Note: *Arm is a registered trademark of Arm limited (or its subsidiaries) in the US and/or elsewhere.*

应用程序独立的软件测试库、自测库，包含在X-CUBE-CLASSB-F4扩展包（与本手册相关联）中的STL_Lib.a文件，版本为V4.0.0。

1.2 参考文件

[1] UM1840: STM32F4系列安全手册，专为面向工业安全的应用程序设计 [2] AN4435: 任何STM32应用程序中获取UL/CSA/IEC 60730-1/60335-1 Class B认证的指南，专为本库的旧版本设计 [3] ES0647: X-CUBE-CLASSB自测库软件勘误表

2 STM32Cube 概览

二点一 STM32Cube是什么？

STM32Cube 是 STMicroelectronics 的一项原创计划，通过减少开发工作量、时间和成本，显著提高设计人员的生产力。STM32Cube 涵盖整个 STM32 产品组合。STM32Cube 包括：

- 一套用户友好的软件开发工具，用于覆盖从构思到实现的项目开发全过程，其中包括：
 - STM32CubeMX，一个图形化软件配置工具，允许通过图形向导自动生成 C 初始化代码 – STM32CubeIDE，一个一体化开发工具，具备外设配置、代码生成、代码编译和调试功能 – STM32CubeCLT，一款集代码编译、板编程和调试功能于一体的一体化命令行开发工具集 – STM32CubeProgrammer (STM32CubeProg)，一款提供图形化和命令行版本的编程工具 – STM32CubeMonitor (STM32CubeMonitor, STM32CubeMonPwr, STM32CubeMonRF, STM32CubeMonUCPD)，强大的监控工具，用于实时精细调整STM32应用的行为和性能 {v*}
- STM32Cube MCU和MPU包，全面的嵌入式软件平台，专为每一系列微控制器和微处理器设计（例如STM32CubeF4用于STM32F4系列），包括：
 - STM32Cube硬件抽象层（HAL），确保在STM32产品系列中实现最大可移植性 – STM32Cube低层API，确保在性能和资源占用方面达到最佳效果，并提供高度的用户对硬件控制 – 一致的中间件组件集合，如RTOS、USB、TCP/IP、图形和FAT文件系统RTOS、USB和图形 – 所有嵌入式软件工具，包含完整的外设和应用示例
- STM32Cube扩展包，其中包含补充STM32Cube MCU和MPU包功能的嵌入式软件组件，包括：
 - 中间件扩展和应用层 – 在某些特定STMicroelectronics开发板上运行的示例

2.2 这款软件如何补充STM32Cube？

软件扩展包通过一个中间件组件扩展STM32Cube，用于管理特定的基于软件的诊断。

该包提供了一个通用的起点，以帮助用户构建和最终确定针对特定应用的安全解决方案。它包含以下内容：

- **STL**：自检库。这提供了一个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，它独立于任何STM32软件运行。它收集微控制器通用组件的自检。
- **用户应用**：这是一个基于一组STM32Cube驱动程序的**STL**集成示例，这些驱动程序通过特定应用测试扩展了**STL**。这部分以完整源代码形式提供，可通过调用终端用户定义的额外特定应用模块进行适应或扩展。该示例可用于库测试，包括对所有提供模块的人工故障支持。

3 STL概述

The **STL** 是由STMicroelectronics发布的一个与应用无关的软件测试库。其目标是提供实现与STM32F4系列微控制器相关的“B类”安全标准所要求的安全机制的相关子集。The **STL** 是一个 **HAL / LL** 独立库，专用于这些微控制器。The **STL** 是一个与编译工具链无关的编译工具，因此任何标准C编译器都可以编译它。The **STL** 是一个自主软件。它根据应用需求执行选定的测试以检测硬件问题，并将结果报告给应用。The **STL** 部分以目标代码形式提供（用于库本身），部分以源代码形式提供（用于用户界面定义和用户参数设置）。

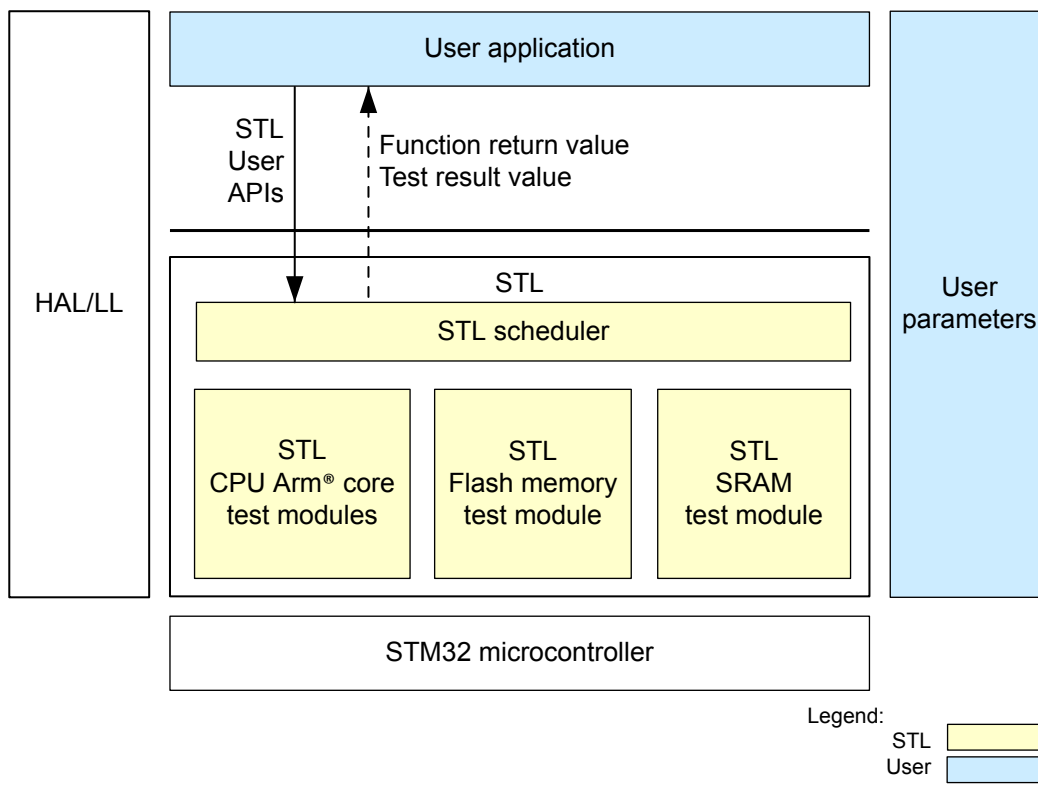
3.1 架构概述

The **STL** 实现了符合 UL/CSA/IEC 60730-1 标准的测试，用于 Arm® Cortex®-M4 **CPU** 核心，以及产品中嵌入的易失性和非易失性存储器。

如下图所示，一个集成**STL**的终端用户应用的系统架构由以下部分组成：

- 用户应用（用浅蓝色表示）
 - 用户参数（以浅蓝色表示）
 - **STL**调度器（以黄色标出）：用户应用可直接通过用户 **API** 访问（不经过 **HAL / LL**）
 - **STL**内部测试模块：由 **STL** 调度程序调用（对用户应用程序不可见）
- 返回给用户应用程序的 **STL** 状态信息在 **API** 层（如表2所示）是：
- 函数返回值收集内部防御性编程检查的结果。
 - 测试模块的结果值存储了测试结果信息。这在一定程度上对应于模块的内部状态（参见第7.3节：状态机）。

图 1. **STL** 架构



STL 还允许开发人员使用人工失败功能。开发人员可以通过强制 **STL** 返回请求的测试结果值来检查应用程序的行为。此功能可通过特定用户API使用。

3.2 支持的产品

The **STL** 运行于STM32F4系列微控制器上，该系列微控制器采用与Cortex®-M4核心及嵌入式存储器相同的架构和集成。

四 STL描述

本节介绍 **STL** 的基础功能和性能信息。本节还概述了终端用户需遵守的限制和强制性操作。

4.1 STL功能描述

一些测试模块可以暂时屏蔽中断。详见第4.2.5节：STL中断屏蔽时间以及第4.3.4节：中断管理。

4.1.1 调度器原理

调度器是用户应用程序所需的**API**模块，以执行**STL**。

主调度器：

- 必须在使用前初始化
- 管理：– 应用的测试模块的初始化和反初始化 – 应用的测试模块的配置 – 应用的测试模块的复位。
- 控制已应用测试序列的执行（**API** 调用）
- 管理"人工故障"用于用户调试和集成测试。
- 通过其特定的校验和确保关键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，无需对 **CPU** 测试模块进行特定的初始化或配置步骤（参见第7.2节：用户**API**和图11）。
- 闪存内存测试操作于定义待测试内存子集的专用用户配置结构的内容（参见第7.1节：用户结构）。这些结构必须由最终用户填写，并在配置和执行闪存内存测试期间保持内容。在执行任何闪存内存测试之前，必须完成测试模块的初始化和配置过程，参见第7.2节：用户**API**和图12。
- **RAM**内存测试操作专用用户配置结构的内容，这些结构定义了待测试内存的子集（参见第7.1节：用户结构）。这些结构必须由最终用户填写，并在**RAM**测试的配置和执行过程中保持内容不变。在执行**RAM**测试之前，必须完成**RAM**测试模块的初始化和配置过程，参见第7.2节：用户**API**和图13。

STL 通过调度器 **API** 被用户在轮询模式下调用。**STL** 可以在中断上下文被调用，但重入是被禁止的。在这种情况下，**STL** 的行为无法得到保证。

用户应用程序必须考虑从**STL**返回的所有信息，该信息通过特定的预定义数据结构提供，该数据结构用于收集状态信息。详见下表。

表 2. STL返回信息

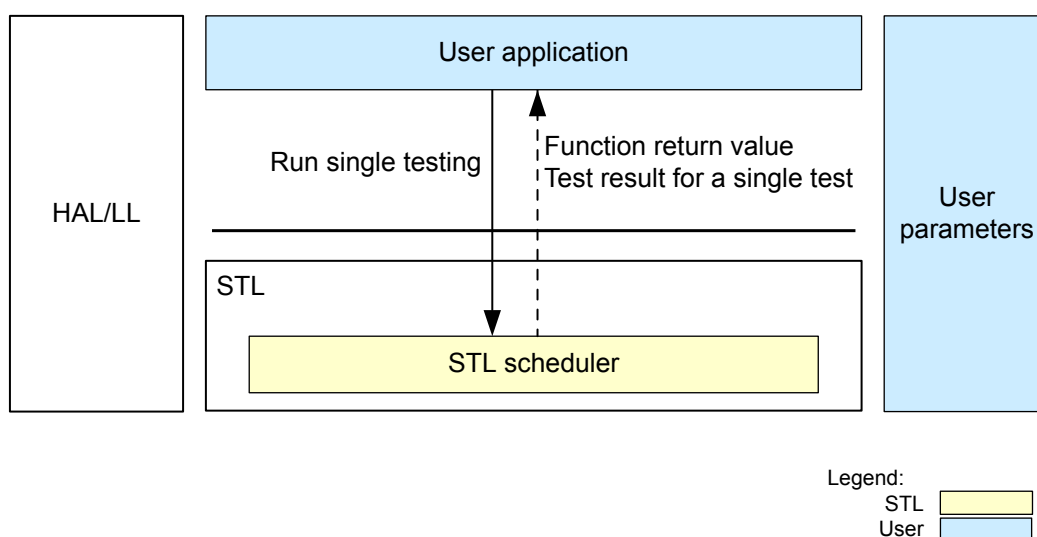
STL information	Value	Description
Function return value ⁽¹⁾	STL_OK	Scheduler function successfully executed
	STL_KO	Scheduler defensive programming error (in this case the test result is not relevant)
Test module result value ⁽²⁾	STL_PASSED	Test passed
	STL_PARTIAL_PASSED	Used only for memory testing when the test passed, but the end of memory configuration has not yet been reached
	STL_FAILED	Hardware error detection by test module
	STL_NOT_TESTED	Test not executed
	STL_ERROR	Test module defensive programming error

1. Refer to `STL_Status_t` definition in Section 7.1: User structures.

2. See `STL_TmStatus_t` in Section 7.1: User structures.

用户应用程序反复应用如下列图所示的调用控制方案，以编程API函数调用序列并管理测试模块的执行顺序。

图2. 单一测试控制调用架构



调度器和中断

调度程序可以随时被中断。

4.1.2

CPU ARM®核心测试

该 **STL** 包含以下列出的 **CPU** 测试模块，以及以下的通用描述（仅作参考信息参考）的测试能力：

- TM1L：实现通用寄存器的轻量模式测试。
- TM7：执行两个堆栈指针的模式和功能测试：MSP，和 PSP
- TMCB：实现对APSR状态寄存器的测试。

注意：

The STL CPU tests are partitioned in separated test modules. This is not intended to allow partial execution of the overall available CPU TMs. It is intended as a support feature to allow better CPU test scheduling in the end-user applications, for example timing constraints. By default, all available TMs are assumed to be executed.

CPU Arm® 核心测试和中断

CPU测试模块可以在任何时间中断。TM7模块仅在最小数据粒度时间内应用屏蔽中断。参见第4.3.4节：中断管理，以获取有关CPU TM7中断管理的详细信息。

4.1.3 闪存测试

原则

的 闪存测试涉及STM32F4的嵌入式闪存存储器 系列。

以下结构必须遵守，以提供闪存存储器测试的正确配置。

- 块：一个连续的4字节区域（FLASH_BLOCK_SIZE），由STL硬编码。
- 章节：一个连续的1024字节区域（FLASH_SECTION_SIZE），由STL硬编码。这与内存物理扇区无关。内存被划分为多个章节。第一个章节从内存的起始地址开始，后续章节彼此连续。用户必须确保在内存完整性测试过程中，对每个需要校验的章节正确计算和放置CRC校验和。
- 二进制（在图4中称为"user program"）：由编译器提供的连续代码区域。它从节的起始位置开始。当二进制区域大小不是节大小的整数倍时，它通常以不完整节结束。在所有情况下，二进制必须是32位对齐的（参见下方的ST CRC 工具信息）。
- 子区域：由用户定义连续区域，包含相邻的节。用户应用程序可以定义一个子区域或多个子区域。子区域必须在二进制区域内定义，其起始地址必须与节的起始位置对齐。它只能包含具有相应预先计算的CRC值的节。当子区域的最后一个节是二进制文件的最后部分时，该节可能不完整。

用户应用程序必须将子集的结束对齐到二进制区域的结束地址。如果仅对一组完整的节进行测试，则子集结束地址必须与最后测试的节的结束对齐。子集大小计算如下：子集大小 = $K * \text{FLASH_SECTION_SIZE} + L * \text{FLASH_BLOCK_SIZE}$ ，其中： $-K$ 是大于0的整数。 $-0 \leq L < (\text{FLASH_SECTION_SIZE} / \text{FLASH_BLOCK_SIZE})$ 当 $L > 0$ 时，二进制的最后一个节是不完整的。用户应用程序定义单个或多个子集及其关联的测试序列。

该 STL 实现了对闪存存储器的测试，以下原理（基于用户配置结构的实际内容）：

- 测试是在由用户应用定义的一个或多个子集的部分上进行的。
- 测试要么连续进行（一次性），要么部分地在单一原子步骤中进行，针对由用户应用程序定义的若干部分。
- 测试结果基于计算的CRC值（在测试执行期间计算）与预期的CRC值（在软件二进制烧录之前计算）之间的CRC比较

执行闪存测试的必经步骤（用于用户应用）是：

- 测试初始化
- 配置一个或多个子集
- 执行测试。

一旦所有子集测试完成，用户需要重置闪存内存测试模块以便重新进行测试。

在STL_ERROR / STL_FAILED测试结果的情况下，测试模块卡在失败的内存子集。在这种情况下，在再次运行之前，请先对闪存内存测试进行反初始化、初始化和重新配置。

预期的CRC预计算

闪存内存测试基于嵌入式硬件CRC计算单元或软件CRC计算算法，可通过一个标志配置。默认配置为CRC硬件单元。要使用软件CRC，必须启用标志STL_SW_CRC，如第5.5.2节中定义的步骤3：从零开始构建应用程序的步骤。CRC是一个32位CRC，符合IEEE 802.3。

闪存存储器的一部分被保留用于**CRC**专用区域，该区域的大小取决于闪存存储器的大小。该区域具有字段格式，其中每个闪存存储器部分都有足够的预留空间以存储一个32位**CRC**模式。用户必须确保所有待测试的区域的字段中都计算并存储有效的**CRC**模式。如图3所示。

每个二进制文件的连续段（从二进制开始到二进制结束）都预先计算了一个期望**CRC**值。这意味着可测试的段数量取决于二进制大小。通常，二进制区域与段大小不对齐。在这种情况下，最后一个不完整段的**CRC**校验值会被预先计算，并且仅在覆盖二进制区域的段部分上进行测试。

前提条件：

- 用户程序区域必须从段的起始位置开始
- 用户程序区域的边界必须32位对齐。
- 根据总闪存内存大小和用户程序大小，最后的程序数据和最初的 **CRC** 数据可能都被存储在同一个闪存区域（无重叠）中。在这种情况下，**CRC** 必须仅基于用户程序数据，参见图4中的示例3。

ST CRC 工具信息

ST 提供了一个 **CRC** 预计算工具。该工具作为 STM32CubeProgrammer（参见第6.2.2节：**CRC** 工具设置）中的一个独立功能存在，它会自动使用填充位（0x00 模式）对二进制文件进行 32 位对齐。

图3. 闪存测试: **CRC** 原理

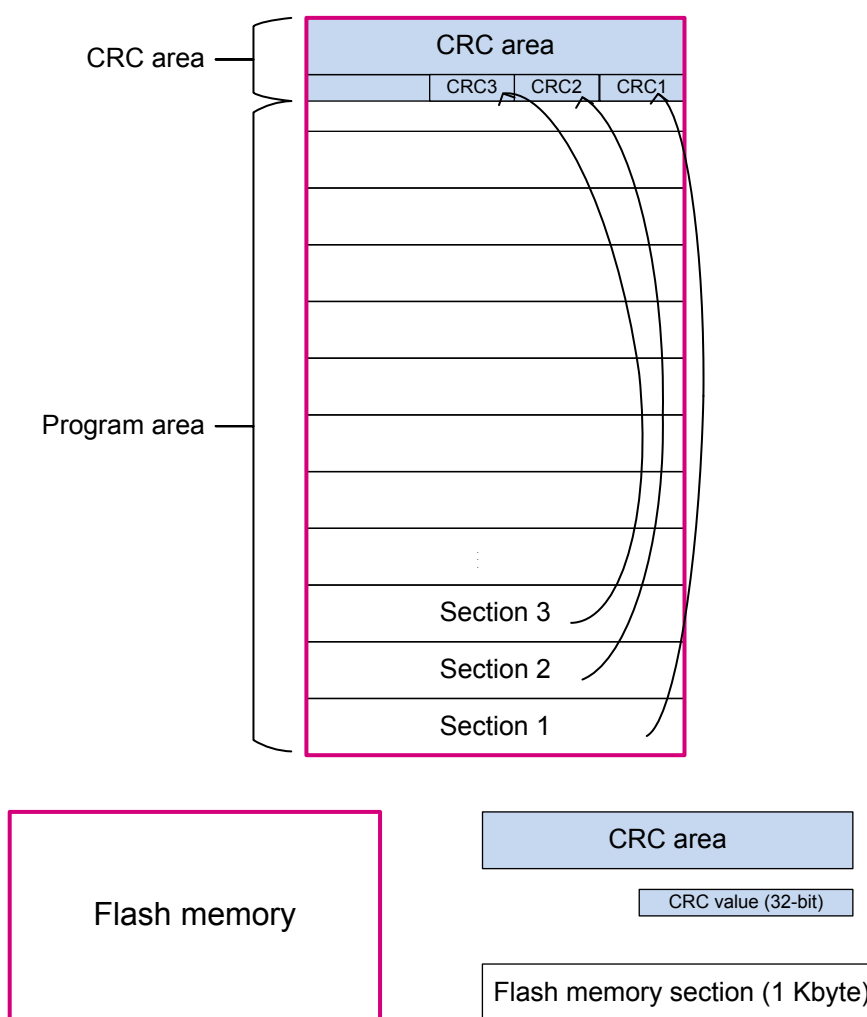
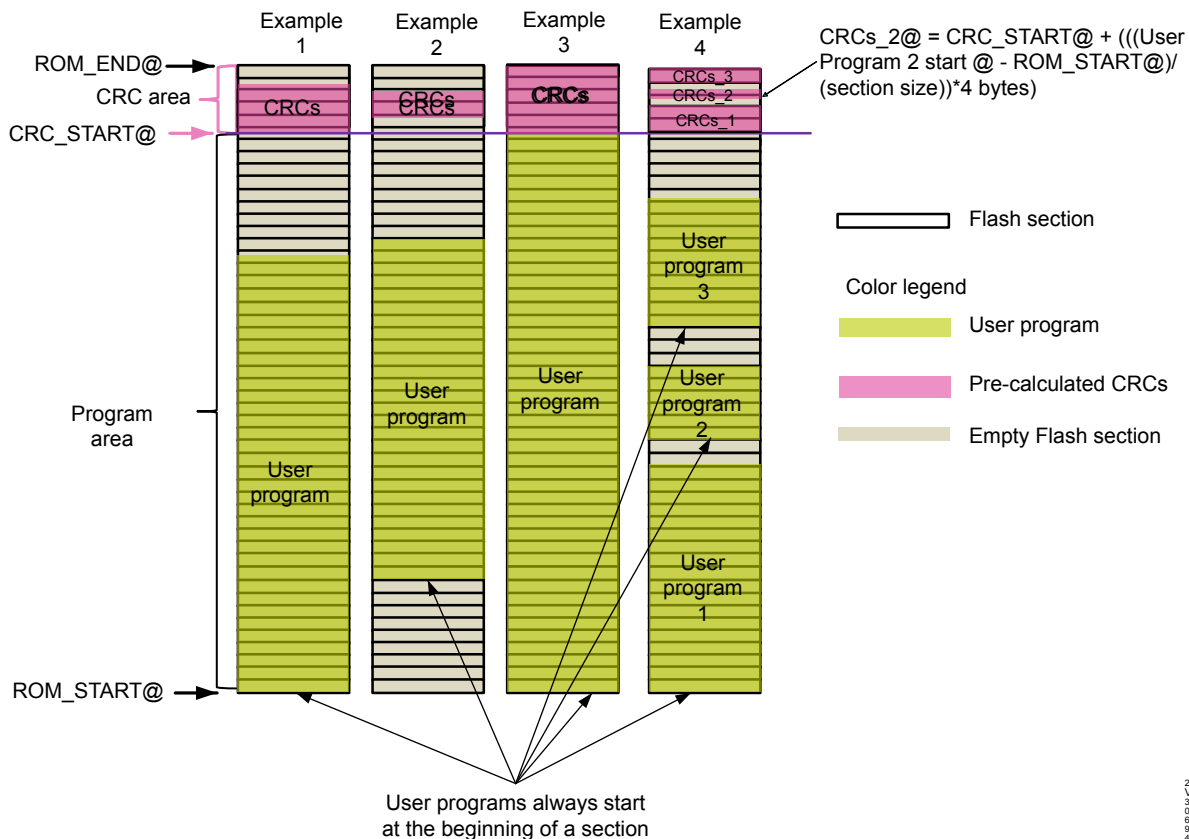


图4. 闪存测试: **CRC** 用例与程序区域



如图4所示的用例描述:

- 示例1: 用户程序从ROM_START地址开始, 因此CRCs从CRC_START地址开始存储。
- 示例2: 用户程序从一个区的起始位置开始, 但不是从ROM_START开始。存储的CRC从CRC区域的正确地址开始。
- 示例3: 用户程序使用整个程序区域, 因此最后的程序数据和第一个CRC数据都存储在同一个内存区域 (无任何重叠)。
- 示例4: 用户程序定义在三个独立区域中。这需要为CRC数据设置三个独立区域。

CRC起始地址计算:

- 实际计算: $\text{CRC起始地址} = (\text{uint32_t})(\text{ROM_END} - 4 * (\text{ROM_END} + 1 - \text{ROM_START}) / (\text{FLASH_SECTION_SIZE}) + 1)$; 其中 FLASH_SECTION_SIZE = 1024
- 文本翻译: $\text{CRC_START} = \text{ROM_END} - (\text{CRC 字节数}) * (\text{内存区的数量}) + 1$

闪存测试和中断

闪存 *TM* 可在任何时间中断。

4.1.4 RAM测试

原则

RAM测试涉及STM32F4系列的嵌入式SRAM存储器。必须遵守以下结构以提供正确的RAM测试配置。

- 块：连续的16字节区域（RAM_BLOCK_SIZE），由 *STL* 硬编码（与内存物理扇区无关）。
- 段：一个连续的128字节区域（RAM_SECTION_SIZE），由 *STL* 硬编码。
- 子集：一个连续区域，其大小是两个块的整数倍，并且起始地址32位对齐。子集的大小不一定是节大小的整数倍，因为子集的最后部分可能小于一个节。
- 子集大小 = $N * \text{RAM_SECTION_SIZE} + 2 * M * \text{RAM_BLOCK_SIZE}$ ，其中：- N 是整数 ≥ 0 - M 是整数 $0 \leq M < 4$ ，当 $M > 0$ 时，最后一个未对齐于段大小的子集的大小

用户应用定义单个或多个子集以及其相关测试序列。

The *STL* 实现了一个RAM内存测试，其原理如下（基于用户配置结构的实际内容）：

- RAM测试在由用户应用程序定义的RAM块上进行
- RAM测试要么一次性进行，要么在用户应用程序定义的若干部分中，通过单个原子步骤分段执行

- 测试实现基于March C-算法

执行RAM测试的强制步骤（针对用户应用）是：

- 初始化RAM测试
- 配置一个或多个RAM子集
- 执行RAM测试

一旦所有子集测试完成，应用程序必须重置RAM测试模块以重新进行测试。在出现STL_ERROR / STL_FAILED测试结果的情况下，测试模块卡在失败的内存子集。在这种情况下，在重新运行测试之前，需要取消初始化、初始化并重新配置RAM。

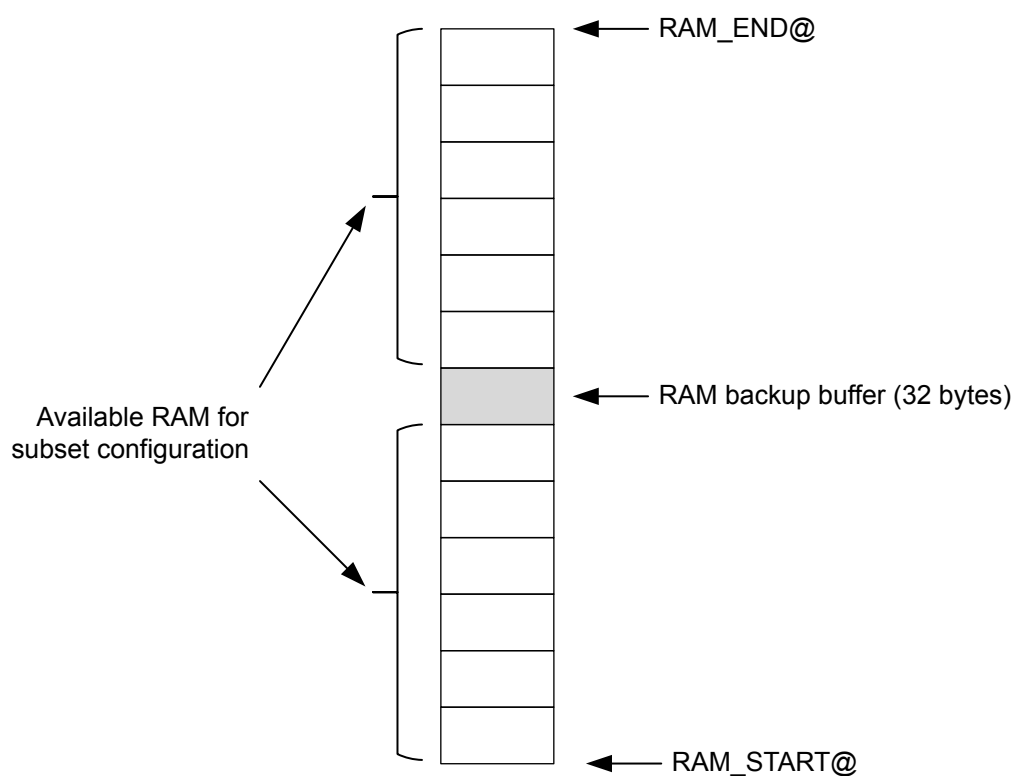
RAM测试和中断

RAM *TM* 可以在任何时间被中断，除了在执行最小数据粒度块时，如第4.2.5节所述：STL中断屏蔽时间。此时，STM32中断和Cortex®具有可配置优先级的异常会被默认暂时屏蔽。请参阅第4.3.4节：中断管理，以获取有关在RAM March-C测试期间中断管理的详细信息。

三月C-测试原理和内存备份原理

RAM测试基于March C-算法，其中内存通过特定模式进行覆盖，然后以特定顺序读取回来。为了恢复初始内存内容，备份过程默认启用并执行。如果不需要，备份过程可以可选禁用。参见第4.3.7节：RAM备份缓冲区，以获取缓冲区控制和分配的详细信息。

图5. RAM测试：使用



4.2 STL性能数据

数据是通过以下测试设置获得的：

- STL库编译细节，描述于应用：编译过程。
- 用于性能测试的项目使用 IAR Embedded Workbench® for Arm® (EWARM) 工具链 v9.40.1 编译
- 编译软件配置，包含：
 - HCLK时钟设置为84 MHz
 - Flash存储器延迟设置为两个等待状态
 - Flash预取功能已启用
 - NULCEO-F429ZI (MB1137 Rev B)

4.2.1 标准模板库执行时间

对应用最优默认 STL 设置时的 STL 执行时间的总结如下面的表格所示。每个 API 的测量数据详见第9节：STL：执行时间详情。

表3. STL执行时间，时钟频率为84 MHz

Tested module	Conditions		Result in μ s
CPU	TM1L, TM7, TMCB		67
Flash memory	Default configuration (STL_SW_CRC not enabled)	1 Kbyte tested	42
		17 Kbytes tested	582
	STL_SW_CRC enabled	1 Kbyte tested	195
		17 Kbytes tested	3183
RAM	Default configuration (neither STL_DISABLE_RAM_BCKUP_BUF, nor STL_ENABLE_IT are enabled)	128 bytes tested	77
		260 Kbytes tested	137484

4.2.2 STL代码和数据大小

STL代码和数据大小详见下表

e.

表4. STL代码大小和数据大小（字节）

Configuration	Module	Flash memory code	Flash memory RO-data	R/W data
STL_SW_CRC not enabled, and STL_DISABLE_RAM_BCKUP_BUF not enabled	stl_user_param_template.o	-	5	44
	stl_util.o	186	-	8
	stl_lib.a	5064	1392	184
STL_SW_CRC enabled, and STL_DISABLE_RAM_BCKUP_BUF not enabled	stl_user_param_template.o	-	2	44
	stl_util.o	94	1	4
	stl_lib.a	4712	1456	184

4.2.3 STL 栈使用

v2 执行可用的 *API* 所需的最小栈可用空间是 200 字节。

4.2.4 STL堆使用

STL 从不进行动态内存分配，因此堆大小与 *STL* 无关。

4.2.5 STL 中断屏蔽时间

STM32中断，以及具有可配置优先级的Cortex®异常，在执行CPU TM7和RAM测试期间被STL 多次屏蔽。如下面的表格所示，RAM测试的最大中断屏蔽时间是取得的。

表5. STL 最大中断屏蔽信息

Tested module	Duration (max) in μ s	Steps
RAM	8	Each execution of STL_SCH_RunRamTM function performs a series of interrupt masking during partial steps of the test at the following time durations: <ul style="list-style-type: none"> 8 μs for backup buffer + 6 μs for the first RAM block to be tested 7 μs for each middle RAM block to be tested⁽¹⁾ 6 μs for the last RAM block to be tested
TM7	5	Masked twice for 5 μ s

1. Number of RAM blocks (multiple of two RAM_BLOCK_SIZE is required) involved with each RAM test execution depends on content of user structures (size of defined subset(s) versus atomic step – see Section 4.1.4: RAM tests)

4.3 STL用户约束

终端用户需要考虑应用程序与STL之间的干扰。忽略此可能导致错误的STL错误报告，以及/或应用程序软件执行问题。因此，为防止任何干扰，应用程序软件与STL集成必须遵守本节中列出的每一项约束。

4.3.1 优先级级别

CPU TM7 和 RAM TM 必须以特权级别执行，以便能够修改某些核心寄存器（例如PRIMASK寄存器），否则这些TMs会返回STL_ERROR。

4.3.2 RCC资源

在STL执行期间，RCC被配置为始终为CRC硬件单元在STL初始化期间以及可选地在STL Flash测试模块执行期间提供时钟。这意味着：

- 当 STL 返回时，它会恢复 CRC 单元的用户 RCC 时钟设置（启用或禁用）
- 用户应用程序在STL执行期间配置RCC时应注意保存/恢复STL设置。

4.3.3 CRC资源

STM32 CRC硬件单元在STL执行期间用于两种不同的情况：

- 在执行 STL 初始化（函数 STL_SCH_Init）时，此阶段中 CRC 硬件单元的使用不能被应用程序软件修改，因此在执行 STL_SCH_Init 函数期间，STL_SW_CRC 标志没有影响。
- 在执行闪存存储器测试模块时，应用程序可以通过STL_SW_CRC标志在硬件和软件计算CRC校验和之间进行选择。默认情况下使用硬件CRC（STL_SW_CRC标志被禁用）。

使用硬件CRC意味着：

- 在调用 STL 之前，用户必须保存任何特定的CRC单元配置。用户配置有在STL执行之后将被恢复。
- 在STL执行期间，硬件单元被配置并用于STL需求（用户应用程序在使用该单元中断STL执行时必须保存/恢复STL设置）。

4.3.4 中断管理

升级机制 - Arm® Cortex® 行为提醒

当 STL 禁用 STM32 中断，并且 Cortex® 可配置优先级的异常时，请注意可能会发生 Arm® Cortex® 升级到 HardFault 的情况。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

默认情况下，在最小数据粒度（几个指令块）内执行 CPU TM7 时，STM32 中断和具有可配置优先级的 Cortex® 异常会被暂时屏蔽，除非用户应用程序激活了 STL_ENABLE_IT 编译开关（参见第5.5.2节：从零开始构建应用程序的步骤）。

如果STL_ENABLE_IT标志被激活，无法保证STL CPU TM7的正确行为。最终用户需负责管理STL与其应用软件之间的干扰，这些干扰可能导致STL生成错误的测试错误报告或无法检测硬件故障。

中断和RAM March C- 测试

默认情况下，在RAM March C-测试期间，STM32中断和具有可配置优先级的Cortex®异常会被屏蔽，除非用户应用程序激活了STL_ENABLE_IT编译开关（参见第5.5.2节：从零开始构建应用程序的步骤）。

如果激活了STL_ENABLE_IT标志：

- 正确的 **STL RAM** 测试行为无法保证，因为应用程序可能在中断处理过程中覆盖 **STL** 测试的 **RAM** 内容。最终用户需负责管理 **STL** 与其应用程序软件之间的干扰，以防止 **STL** 生成错误的 **RAM** 测试错误报告。
- 用户应用程序软件的行为可能被破坏。错误数据可能被从 **RAM** 正在被 **STL** 三月 C-测试修改的位置。

中断和通用寄存器

在 **STL** 执行期间，应用程序必须保存和恢复通用寄存器，以在 STM32 中断和 Cortex® 异常的可配置优先级服务例程中确保正确的 **STL** 行为并防止任何误报错误。

如何 **STL** 屏蔽中断

为了屏蔽中断，**STL** 将 PRIMASK 寄存器位设置为 1。将此位设置为 1 会将当前执行优先级提升至 0，防止激活所有具有可配置优先级的异常。因此，当当前执行优先级被提升至特定值时，所有具有较低或相等优先级的异常均被屏蔽。

4.3.5 直接内存访问

应用程序必须管理 DMA，以避免在 RAM 银行期间的未授权访问，**STL** March C-测试。在这种情况下：

- DMA 写入可能会干扰 **STL** 测试，导致错误的错误报告
- DMA 读取可能会返回错误数据，由于 **STL** 对 DMA 专用 RAM 区域的覆盖写入。

4.3.6 支持的内存

提供的 **STL** 内存测试（Flash™ 和 RAM™）必须仅在 STM32 内部嵌入式内存上执行。提供的 **STL** 库流程必须仅从内部嵌入式内存执行。

4.3.7 RAM 备份缓冲区

备份过程在 RAM 测试期间默认启用以保留 RAM 内容。用户必须在编译时保留特定区域作为 RAM 备份缓冲区，该区域必须在 RAM 子集配置之外分配。为测试仅定义了一个 RAM 备份缓冲区。每次调用 RAM 运行测试时，RAM 备份缓冲区区域也会通过 March C-算法进行测试（在用户定义的任何子集被测试之前）。

备份过程可以通过激活编译开关 **STL_DISABLE_RAM_BCKUP_BUF**（参见第 5.5.2 节：从零开始构建应用程序的步骤）永久禁用，或者通过特定的控制序列暂时抑制（参见下方注释）。在这种情况下，最终用户需确保应用程序软件不会消耗 March C 测试中被破坏的数据。此选项可用于加速那些用户不需要保留内存内容的 RAM 区域的测试。

Note:

For a temporary suppression of the RAM backup buffer, the user must follow a set sequence:

1. *Change the **STL_pRamTmBckUpBuf** variable value to overwrite it by NULL, while keeping a backup of the original value (default value stored by the STL).*
 2. *The RAM test must then be restarted. To do so the user can use one of the APIs which force the RAM test into either **RAM_IDLE** or **RAM_INIT** state (see state diagrams in Section 7.3: State machines) .*
- To remove the backup suppression, the user must perform the same steps as above while restoring the default value of **STL_pRamTmBckUpBuf** to its original value and reinitialize the RAM test.*

4.3.8 内存映射

由于 RAM 测试模块和 March C 方法设计，用户必须确保 **STL** 的“只读”数据位于闪存存储器中。这必须通过相关链接器文件的适当调整来完成。以下示例适用于 EWARM 和 STM32CubeIDE。

EWARM .icf 文件适配示例

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

STM32CubeIDE .ld 文件自定义示例

```
...} >只读存储器
```

Note:

Usually the default configuration locates "read only" data in flash memory.

4.3.9 处理器模式

The **STL CPU** TM7 must be executed in thread mode in order to set the active stack pointer to the process stack pointer. If the **STL** is not executed in thread mode, the **CPU** TM7 returns STL_ERROR.

4.4 终端用户集成测试

本节描述了终端用户在验证阶段必须执行的强制性测试。这些测试确保 **STL** 正确集成在应用程序软件中。

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）来检查每个计划的诊断功能是否已正确执行。这涉及测试模块的执行及其所有配置操作。

4.4.2 测试2: 修复STL错误信息处理

终端用户必须检查由 **STL** 函数返回和测试模块结果值产生的任何错误信息是否被正确解释为异常行为，并在其应用程序中正确处理。错误信息指与预期值不同的值，参见第7.2节：用户API。在验证过程中，必须使用人工故障功能来模拟生成与相关个别软件诊断（**CPU** 测试，**RAM** 测试，闪存内存测试）相关的错误测试模块结果值，针对每个使用的个别功能。

此过程不能被视为对实际 **CPU** 故障在真实设备上的全面模拟，而更像已实现的 **API** 的测试接口。

Note:

In some circumstances, experts performing the safety assessment of final systems embedding the STL might require exhaustive simulation to demonstrate STL capability to capture corruption of STM32 registers or memories injected intentionally during debugging the STL code. To perform these tests is practically impossible for any end user due to STL object delivery format. This specific testing was done for all the provided TMs during the STL certification process and its passing is recorded at internal test reports and guaranteed by the valid certificate issued for this ST firmware.

五 软件包描述

本节详细说明了 X-CUBE-CLASSB-F4 扩展包内容及其正确使用方法。

5.1 一般描述

X-CUBE-CLASSB-F4 是用于 STM32F4 系列微控制器的软件扩展包。它提供了一套完整的解决方案，帮助终端客户构建安全应用：

- 一个与应用程序无关的软件测试库可供使用：
 - 部分作为目标代码：STL_Lib.a，库本身 – 部分作为源文件：stl_user_param_template.c 和 stl_util.c
 - 包含三个头文件：stl_stm32_hw_config.h、stl_user_api.h 和 stl_util.h

- 一个用户应用程序示例，可作为源代码获取。

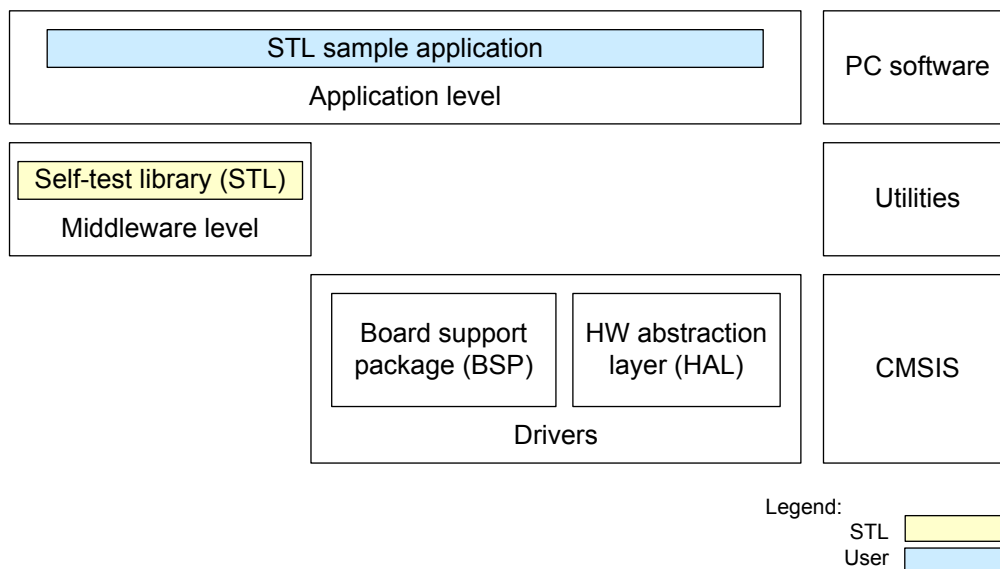
X-CUBE-CLASSB-F4 已移植到第 3.2 节列出的受支持产品中。软件扩展包包含一个示例应用程序，开发者可以使用它来开始对代码进行实验。它以包含源代码和库的 zip 存档形式提供。以下集成开发环境受支持：

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

5.2 架构

X-CUBE-CLASSB-F4 扩展包的组件如图 6 所示。

图 6. 软件架构概览



5.2.1 STM32Cube HAL

The **HAL** 驱动层提供一个简单、通用、多实例的 **API** 集合（应用程序编程接口），以与上层（应用、库和堆栈）进行交互。

它包括通用和扩展 **API** 架构。它直接基于通用架构构建，并允许其上构建的层（例如中间件层）实现其功能，而无需依赖特定微控制器的硬件配置。

这种结构 improves 库代码的可重用性，并保证易于移植

与其他设备的兼容性

5.2.2 板级支持包 (BSP)

该软件包需要支持STM32开发板上的外设，除了MCU。该软件包含在板级支持包（BSP）中。这是一个有限的API集合，为某些特定的板级组件（如LED灯和用户按钮）提供编程接口。

5.2.3 标准模板库

在中间件层中可获得的STL的一个重要部分是一个管理基于软件的诊断测试的黑箱。它独立于HAL、BSP和CMSIS，即使STL的集成示例依赖于某些HAL驱动程序。

5.2.4. 用户应用示例

示例中提供的参考项目展示了如何在采用不同IDE时，将可能的STL测试模块调用序列集成到应用程序中，验证API的返回值，并人工模拟其失败响应。此外，通过应用监控方法对时钟系统进行测试的专用模块

符合“Class B”标准要求的包含完整的源代码以扩展可用的库集合。它展示了库如何通过由最终用户完全定义的特定测试或模块进行扩展。

该示例还展示了一种可能的方法，用于区分整体初始启动测试与在应用程序运行期间定期执行的部分测试序列。

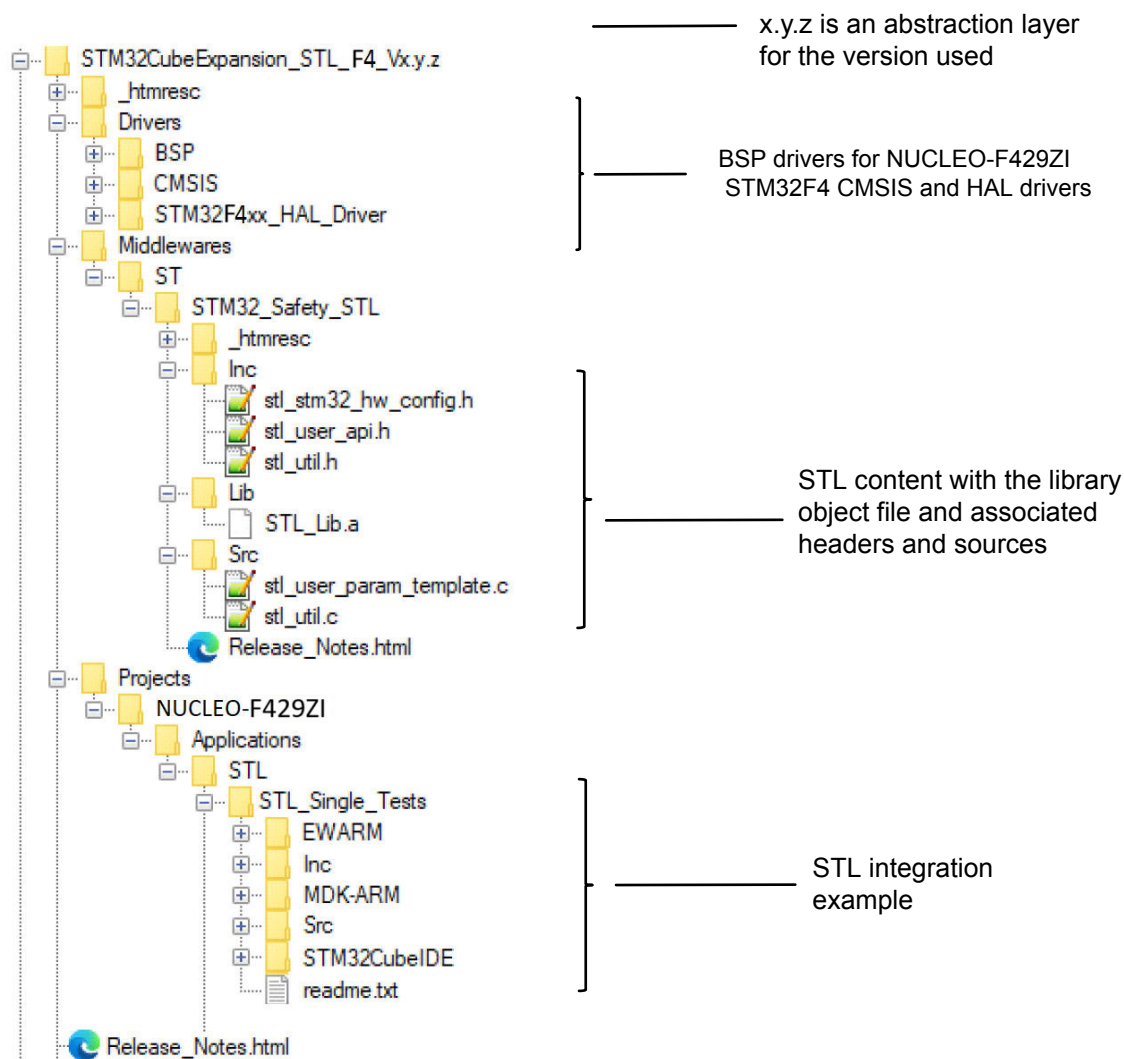
5.2.5 STL完整性

STL内容的完整性通过哈希SHA-256确保。

5.3 文件夹结构

如图7所示，结构的顶层视图。

Figure 7. Project file structure



1
V
0
0
9
7
7
D

5.4 应用程序编程接口

5.4.1 合规

接口符合性

库部分的 *STL*，未包含在源代码中，已使用 IAR Embedded Workbench® for Arm v9.30.1 编译。编译过程中使用了 `--aeabi` 和 `--guard_calls` 编译选项，以满足 AEABI 合规性，如 EWARM 帮助部分的“AEABI compliance”所述。

为符合IEC 61508功能安全标准，该自检库（X-CUBE-STL）的工业版本经TUV认证，仅由安全认证的IAR™编译器EWARMFS版本编译。此X-CUBE-CLASSB自检库的源代码专用于家电，基于工业版本库中的一组模块，并已进行调整以符合IEC 60730-1标准。此库可以与任何标准C编译器链接。

安全指南

为符合如IAR Embedded Workbench®安全指南（指南中的建议2.1-1、2.2-5、2.4-1a和5.4-3）及Keil®安全手册（§4.9.2）所述的安全指南，采用--strict、--remarks、--require_prototypes和--no_unaligned_access编译选项来实现符合性。

图书馆合规

库部分的 **STL**（未包含在源代码中）符合ISO C99标准库。它使用IAR™选项进行编译。C语言方言 = 标准C。

Arm® 编译器C工具链 厂商/版本独立性

The **STL**用户 **API** 仅指 C语言中的“uint32_t”和“enum”类型：

- “uint32_t” C类型根据C99标准具有固定的32位类型大小。
- “枚举”类型大小，根据C标准C99，由实现定义。它必须能够表示所有枚举成员的值。在**STL**接口中，枚举类型值为无符号整数，小于或等于(232 - 1)。用户必须确保枚举类型值能够容纳32位值。

五.四.二 依赖{v*}

The **STL** library 调用 `memset` 标准 C 库函数。

此外，还使用IAR™ EWARM工具链编译器来编译 **STL** 库。该编译器可能在某些情况下调用以下标准C库函数：`memcpy`、`memset`、和 `memclr`。这种行为是IAR™ EWARM工具链编译器的固有特性。无法禁用或避免。

因此，在链接 **STL** 库时，用户必须确保这些标准 C 库函数被定义。用户可以使用工具链提供的函数或自定义的函数。

5.4.3 详情 {v*}

有关可用的 **API** 的详细技术信息，请参见第7.2节：用户API，其中描述了函数和参数。

5.5 应用：编译过程

5.5.1 构建提供的STL示例的步骤

1. 安装**ST CRC**工具（参见第6.2.2节：**CRC**工具设置）或其他**CRC**工具生成足够的执行闪存测试所需的结构。2. 项目选择：选择一个项目示例并打开它。3. 项目构建：启动构建，该构建会编译二进制文件，并在构建后命令中调用 **CRC** 工具以计算和分配 **CRC** 结果。出现错误时，请检查 **CRC** 工具路径。详情请参见 第5.5.2节：从零开始构建应用程序的步骤。4. 加载编译后的二进制文件。5. 执行。

启动板卡并检查结果：

- LED快速且规律地切换：测试结果符合预期。
- LED异常切换：存在错误。

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

则调用 FailSafe_Handler 过程，该过程使用一个参数保存失败模块的识别码

Note: *The codes definitions are given in the `stl_user_api.h` file, in the case of a defensive programming failure, the `DEF_PROG_OFFSET` is added to the module code. User can adapt or extend the set of definitions applied by the STL example there.*

5.5.2 从零开始构建应用程序的步骤

要从头开始构建应用程序，请按照以下步骤操作：1. 创建一个具有合适目录结构和所有适当包的新应用程序项目。使用STM32CubeMX工具自动完成此操作。2. 如果项目中任何STL的自动包含选项不被STM32CubeMX工具支持，请将交付的STL 示例中的...Middleware\ST\STM32_Safety_STL目录内容复制并粘贴到应用程序项目的目录结构中。参见第5.3节：文件夹结构。在这种情况下，按照以下步骤手动修改项目设置：– 将Src目录中的所有STL源文件添加到项目中。– 将Inc目录设置为项目设置中的附加包含路径。– 强制链接器将Lib目录中的库对象文件作为附加库包含。

Note: *This second step is necessary only when no automatic including option is supported by the CubeMX tool else it is fully performed by the tool - then there is no need for any manual intervention as described above - user can leave them out and continue by Step 3*

3. 如有需要，在项目设置中添加以下可选预处理器编译开关：– 启用STL_DISABLE_RAM_BCKUP_BUF选项，如果未使用RAM备份缓冲区（在此情况下，被测试子集的RAM数据将被销毁）。若未激活，将默认使用RAM备份缓冲区。在这种情况下，必须在链接器分散文件中定义"backup_buffer_section"段。– 启用STL_SW_CRC选项：这是用户应用程序选择软件CRC的位置。若未激活，将默认使用硬件CRC计算。– 启用STL_ENABLE_IT选项：这是用户应用程序在CPU TM7和RAM测试期间启用STM32中断的位置。若未激活，这些测试期间将屏蔽中断。参见第4.3.4节：中断管理，以及第4.1.4节：RAM测试。4. 检查闪存存储器密度的配置。必须在stl_user_param_template.c文件中为项目设置正确的内存范围。特别更新STL_ROM_END_ADDR，以确保与相关链接器分散文件的一致性。

并且CRC工具脚本（参见步骤6。）。5. 开发用户STL流控制。这是通过实现正确的API调用序列，在周期性循环中重复，如定义的安全任务要求。必须确保正确填充所有相关用户结构，以控制内存测试并应用正确的STL返回信息检查。参见第7节：STL：用户API和状态机。6. 应用CRC工具构建CRC区域内容，以满足CRC计算需求。参见第6.2.2节：CRC工具设置。执行STM32CubeProgrammer的正确命令行。这可以在编译过程中通过调用IDE的后构建功能操作自动完成，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，可以使用人工故障API来调试编程STL流的正确行为。

图8. IAR™构建后操作截图

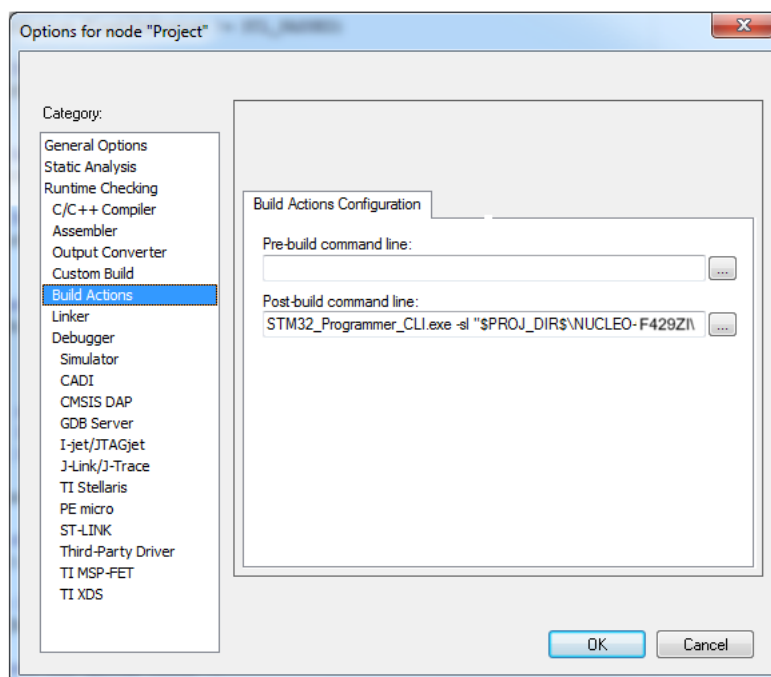
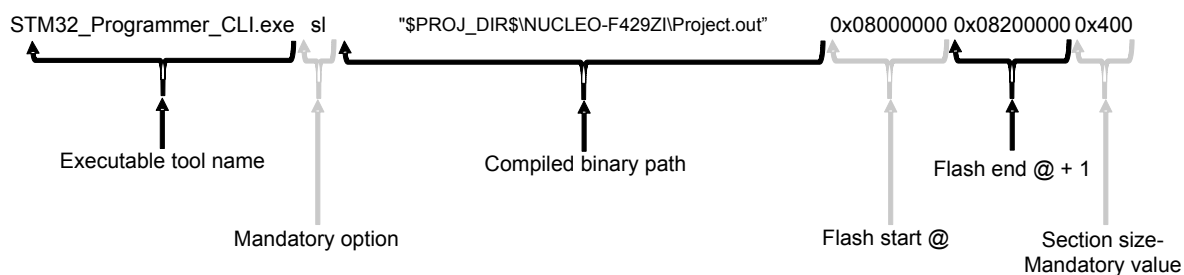


图9. CRC 工具命令行



1
V
1
0
0
7
T
D

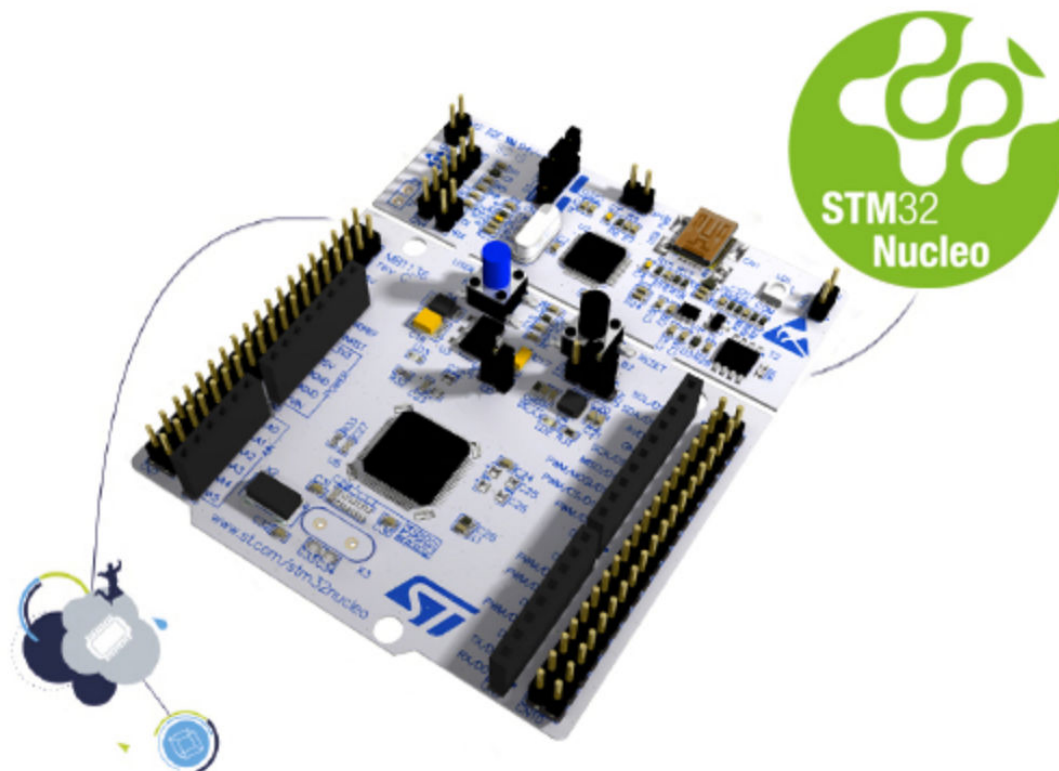
六 硬件和软件环境配置

6.1 硬件设置

STM32 Nucleo开发板为用户提供了一种经济实惠且灵活的方式，可以尝试新创意并使用任何STM32微控制器系列构建原型。ARDUINO®连接支持和ST morpho接头使得通过广泛的专用扩展板轻松扩展STM32 Nucleo开放开发平台的功能。STM32 Nucleo开发板无需单独的探针，因为它集成了ST-LINK/V2-1调试器/编程器。STM32 Nucleo开发板附带STM32全面的软件HAL库以及各种封装软件示例。关于STM32 Nucleo的详细信息

开发板可在以下网址获取：<http://www.st.com/stm32nucleo> 网页。

图10. STM32 Nucleo开发板示例



以下组件是必需的:

- NULCEO-F429ZI (MB1137 B版) 开发板
- USB A型到Mini-B型USB电缆，用于将开发板连接到PC。

6.2 软件设置

本节列出了开发人员设置SDK、运行示例场景和自定义应用程序所需的最低要求。

6.2.1 开发工具链和编译器

从 STM32Cube 软件扩展包支持的 *IDE* 中选择一个。阅读所选 *IDE* 提供商提供的系统要求和设置信息。检查发布包中 projects 目录下的 Release_Notes.html 文件，并参考是否存在章节 *IDE* 兼容性。

6.2.2 **CRC** 工具设置

ST 提供了一个 **CRC** 工具，作为 STM32CubeProgrammer 内的一个独立功能，用于闪存存储器测试。其他 **CRC** 工具也可以使用，前提是它们满足 Expected CRC 预计算中详细说明的要求。

工具安装步骤:

1. 在 www.st.com 上找到的专用网页上选择 STM32CubeProgrammer 2. 安装该包。最简单的方法是将工具路径添加到环境变量中（需要计算机管理员权限）。否则，必须直接在项目中添加路径用于编译，在构建后选项中。

七 STL: 用户API和状态机

7.1 用户结构

The structures are defined in `stl_user_api.h`. It is forbidden to change the content of this file.

Structures detailed hereafter are copies of the `stl_user_api.h` content:

```
typedef enum
{
    STL_OK = STL_OK_DEF, /* Scheduler function successfully executed */
    STL_KO = STL_KO_DEF /* Scheduler function unsuccessfully executed
                        (defensive programming error, checksum error). In this case
                        the STL_TmStatus_t values are not relevant */
} STL_Status_t; /* Type for the status return value of the STL function execution */
```

```
typedef enum
{
    STL_PASSED = STL_PASSED_DEF, /* Test passed. For Flash/RAM, test is passed and end of
                                configuration is also reached */
    STL_PARTIAL_PASSED = STL_PARTIAL_PASSED_DEF, /* Used only for RAM and Flash testing.
                                                Test passed, But end of Flash/RAM
                                                configuration not yet reached */
    STL_FAILED = STL_FAILED_DEF, /* Hardware error detection by Test Module */
    STL_NOT_TESTED = STL_NOT_TESTED_DEF, /* Initial value after a SW init, SW config,
                                        SW reset, SW de-init or value when Test Module
                                        not executed */
    STL_ERROR = STL_ERROR_DEF /* Test Module unsuccessfully executed (defensive programing
                                check failed) */
} STL_TmStatus_t; /* Type for the result of a Test Module */
```

```
typedef enum
{
    STL_CPU_TM1L_IDX = 0U, /* CPU Arm Core Test Module 1L index */
    STL_CPU_TM7_IDX, /* CPU Arm Core Test Module 7 index */
    STL_CPU_TMCB_IDX, /* CPU Arm Core Test Module Class B index */
    STL_CPU_TM_MAX /* Number of CPU Arm Core Test Modules */
} STL_CpuTmIndex_t; /* Type for index of CPU Arm Core Test
                    Modules */
```

```
typedef struct STL_MemSubset_struct
{
    uint32_t StartAddr; /* start address of Flash or RAM memory subset */
    uint32_t EndAddr; /* end address of Flash or RAM memory subset */

    struct STL_MemSubset_struct *pNext; /* pointer to the next Flash or RAM memory subset
                                        - to be set to NULL for the last subset */
} STL_MemSubset_t; /* Type used to define Flash
                    or RAM subsets to test */
```

```
typedef struct
{
    STL_MemSubset_t *pSubset; /* Pointer to the Flash or RAM subsets to test */
    uint32_t NumSectionsAtomic; /* Number of Flash or RAM sections to be tested
                                during an atomic test */
} STL_MemConfig_t; /* Type used to fully define Flash or RAM test configuration */
```

```
typedef struct
{
    STL_TmStatus_t aCpuTmStatus[STL_CPU_TM_MAX]; /* Array of forced status value
                                                for CPU Test Modules */
    STL_TmStatus_t FlashTmStatus; /* Forced status value for Flash Test Module */
    STL_TmStatus_t RamTmStatus; /* Forced status value for RAM Test Module */
} STL_ArtifFailingConfig_t; /* Type used to force Test Modules status to a specific
                             value for each STL Test Module */
```

7.2 用户API

以下 API 在文件 `stl_user_api.h` 中声明。禁止修改此文件的内容。

注意:

For pointers defined by the user application and used as STL API parameters, the user application must set valid pointers, maintain pointer availability, and check the pointer integrity. The STL does not copy the pointer content, and accesses directly to the memory addresses defined by the application.

This applies during the overall STL execution. For example, the pointers to access the content of structures that keep the configuration of the memory tests must be maintained. They are still used by the `STL_SCH_run_xxx` functions, even if they are not always part of the input parameter list when an API associated with these tests is called.

For more details about proper API sequence calls see Section 7.3: State machines and Section 8: Test examples.

7.2.1 通用API

以下章节提供了关于常见 APIs 的详细信息。

7.2.1.1 **STL_SCH_Init**

描述: 初始化调度器。可以在任何时间调用以重新初始化调度器（重置所有测试）。声明: `STL_Status_t STL_SCH_Init(void)`。

表6. STL_SCH_Init输入信息

Allowed states	Parameters
CPU TMx: all Flash TM: all RAM TM: all	-

表7. STL_SCH_Init输出信息

STL_Status_t return value		Returned state
Value	Comments	
STL_OK	Function successfully executed	CPU TMx: CPU_TMx_CONFIGURED Flash TM: FLASH_IDLE RAM TM: RAM_IDLE
STL_KO	Source of defensive programming error: • STL internal data corrupted	No state change

附加信息: 对于 CPU 测试模块, 没有特定的 CPU 初始化函数。

Note:

This function uses hardware CRC as explained in Section 4.3.3: CRC resources.

7.2.2 CPU ARM® 核心测试 API

7.2.2.1 **STL_SCH_RunCpuTMx**

描述: 运行其中一个 CPU 测试模块。

声明: `STL_Status_t STL_SCH_RunCpuTMx(STL_TmStatus_t *pSingleTmStatus)` 其中 TMx 可以是 TM1L、TM7 或 TMCB 中的一种。

表8. STL_SCH_RunCpuTMx 输入信息

Allowed states	Parameters	
	Value	Comments
CPU_TMx_CONFIGURED	*pSingleTmStatus	See Caution

表9. STL_SCH_RunCpuTMx 输出信息

STL_Status_t return value		*pSingleTmStatus output		Returned state
Value	Comments	Value	Comments	
STL_OK	Function successfully executed	STL_PASSED	-	CPU_TMx_CONFIGURED
		STL_FAILED	-	
		STL_ERROR	Source of defensive programming error: <ul style="list-style-type: none"> STL internal data corrupted Software is not executed with privileged level for CPU TM7 Software is not executed in thread mode for CPU TM7 	
STL_KO	Possible source of defensive programming error: <ul style="list-style-type: none"> pSingleTmStatus = NULL STL internal data corrupted 	Not relevant	Value must not be used	No state change

第7.2.3节 测试API

7.2.3.1 STL_SCH_InitFlash

描述：启动闪存测试。

声明：STL_Status_t STL_SCH_InitFlash(STL_TmStatus_t *pSingleTmStatus)

表 10. STL_SCH_InitFlash 输入信息

Allowed states	Parameters	
	Value	Comments
FLASH_IDLE FLASH_INIT FLASH_CONFIGURED	*pSingleTmStatus	Caution

表 11. STL_SCH_InitFlash 输出信息

STL_Status_t return value		*pSingleTmStatus output		Returned state
Value	Comments	Value	Comments	
STL_OK	Function successfully executed	STL_NOT_TESTED	-	FLASH_INIT
STL_KO	Possible source of defensive programming error: <ul style="list-style-type: none"> pSingleTmStatus = NULL STL internal data corrupted 	Not relevant	Value must not be used	No state change

7.2.3.2 STL_SCH_ConfigureFlash

描述：配置闪存内存测试。

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

Allowed states	Parameter		
	Value	Comments	
FLASH_INIT	*pSingleTmStatus	See Caution	
	*pFlashConfig	Pointer to the flash memory configuration. See Caution.	
		Field	Comments
		*pSubset	<ul style="list-style-type: none"> Pointer to flash memory subset. See Caution A section cannot overlap with the CRC area
		Field	Comments
		StartAddr	<ul style="list-style-type: none"> Start subset address in bytes Cannot be lower than ROM_START and higher than CRC_START address
		EndAddr	<ul style="list-style-type: none"> End subset address in bytes Cannot be lower than ROM_START and higher than CRC_START address Needs to be higher than StartAddr
		*pNext	<ul style="list-style-type: none"> Pointer to next flash memory subset. See Caution Must be set to NULL for the last subset
		NumSectionsAtomic	<ul style="list-style-type: none"> Number of flash memory sections to be tested during an atomic test Set to 1, as minimum (one section per test) If the value is higher than the number of sections in all subsets, all flash memory subsets are tested in one pass

表 13. STL_SCH_ConfigureFlash 输出信息

STL_Status_t return value		*pSingleTmStatus output		Returned state
Value	Comments	Value	Comments	
STL_OK	Function successfully executed	STL_NOT_TESTED	-	FLASH_CONFIGURED
		STL_ERROR	Possible source of defensive programming error: <ul style="list-style-type: none"> State not allowed Wrong configuration detected STL internal data corrupted 	No state change
STL_KO	Possible source of defensive programming error: <ul style="list-style-type: none"> pSingleTmStatus = NULL pFlashConfig = NULL STL internal data corrupted 	Not relevant	Value must not be used	No state change

附加信息：当返回值设置为 STL_KO 或 *pSingleTmStatus 设置为 STL_ERROR 时，闪存内存配置不会被应用。

7.2.3.3

STL_SCH_RunFlashTM

描述：执行闪存内存测试。

声明：STL_Status_t STL_SCH_RunFlashTM(STL_TmStatus_t *pSingleTmStatus)

表 14. STL_SCH_RunFlashTM 输入信息

Allowed states	Parameters	
	Value	Comments
FLASH_CONFIGURED	*pSingleTmStatus	See Caution

表15. STL_SCH_RunFlashTM输出信息

STL_Status_t return value		*pSingleTmStatus output		Returned state
Value	Comments	Value	Comments	
STL_OK	Function successfully executed	STL_PASSED	-	FLASH_CONFIGURED
		STL_PARTIAL_PASSED	-	FLASH_CONFIGURED
		STL_FAILED	-	FLASH_CONFIGURED
		STL_NOT_TESTED	All subsets are already tested	FLASH_CONFIGURED
		STL_ERROR	Possible source of defensive programming error: <ul style="list-style-type: none"> State not allowed Configuration corrupted STL internal data corrupted 	No state change
STL_KO	Possible source of defensive programming error: <ul style="list-style-type: none"> pSingleTmStatus = NULL STL internal data corrupted 	Not relevant	Value must not be used	No state change

7.2.3.4
STL_SCH_ResetFlash

描述: 重置闪存内存测试。

声明: STL_Stat us_t STL_SCH_ResetFlash(STL_TmStatus_t *pSing leTm状态)

表16. STL_SCH_ResetFlash 输入信息

Allowed states	Parameters	
	Value	Comments
FLASH_CONFIGURED	*pSingleTmStatus	See Caution

表17. STL_SCH_ResetFlash输出信息

STL_Status_t return value		*pSingleTmStatus output		Returned state
Value	Comments	Value	Comments	
STL_OK	Function successfully executed	STL_NOT_TESTED	Configuration successfully applied	FLASH_CONFIGURED
		STL_ERROR	Possible source of defensive programming error: <ul style="list-style-type: none"> State not allowed Configuration corrupted STL internal data corrupted 	No state change
STL_KO	Possible source of defensive programming error: <ul style="list-style-type: none"> pSingleTmStatus = NULL STL internal data corrupted 	Not relevant	Value must not be used	No state change

附加信息

- 在所有子集测试完成后，用户需要重置测试模块以再次执行闪存测试。
- 当返回值被设置为 STL_KO 或 *pSingleTmStatus 被设置为 STL_ERROR 时，闪存复位不会被应用。

7.2.3.5 STL_SCH_DeInitFlash

描述：反初始化闪存测试。

声明：STL_Status_t STL_SCH_DeInitFlash(STL_TmStatus_t *pSingleTmStatus)

表 18. STL_SCH_DeInitFlash 输入信息

Allowed states	Parameters	
	Value	Comments
FLASH_IDLE FLASH_INIT FLASH_CONFIGURED	*pSingleTmStatus	See Caution

表 19. STL_SCH_DeInitFlash 输出信息

STL_Status_t return value		*pSingleTmStatus output		Returned state
Value	Comments	Value	Comments	
STL_OK	Function successfully executed	STL_NOT_TESTED	-	FLASH_IDLE
STL_KO	Possible source of defensive programming error: <ul style="list-style-type: none"> pSingleTmStatus = NULL STL internal data corrupted 	Not relevant	Value must not be used	No state change

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

Allowed states	Parameters	
	Value	Comments
RAM_IDLE RAM_INIT RAM_CONFIGURED	*pSingleTmStatus	See Caution

表21. STL_SCH_InitRam输出信息

STL_Status_t return value		*pSingleTmStatus output		Returned state
Value	Comments	Value	Comments	
STL_OK	Function successfully executed	STL_NOT_TESTED	-	RAM_INIT
STL_KO	Possible source of defensive programming error: <ul style="list-style-type: none"> pSingleTmStatus = NULL STL internal data corrupted 	Not relevant	Value must not be used	No state change

7.2.4.2 STL_Status_t STL_SCH_ConfigureRam

描述: 描述: 配置RAM测试。

声明: STL_Status_t STL_SCH_ConfigureRam(STL_TmStatus_t *pSingleTmStatus, STL_MemConfig_t 类型的指针 pRamConfig)

表22. STL_SCH_ConfigureRam 输入信息

Allowed states	Parameter	
	Value	Comments
RAM_INIT	*pSingleTmStatus	See Caution
	*pRamConfig	This pointer contains the RAM configuration. See Caution
		Field
		Comments
		<ul style="list-style-type: none"> Pointer to RAM subset. See Caution A subset cannot overlap with the RAM backup buffer if defined
		Field
		Comments
	*pSubset	StartAddr
		<ul style="list-style-type: none"> Start subset address in bytes Start address must be 32-bit aligned RAM subset must be inside RAM area Cannot be lower than RAM_START and higher than RAM_END address
		EndAddr
		<ul style="list-style-type: none"> End subset address in bytes Higher than StartAddr Cannot be lower than RAM_START and higher than RAM_END address Subset size (EndAddr-StartAddr) needs to be multiple of 2 * RAM_BLOCK_SIZE, 32 bytes
	*pNext	<ul style="list-style-type: none"> Pointer to next RAM subset. See Caution Must be set to NULL for the last subset
		NumSectionsAtomic
		<ul style="list-style-type: none"> Number of RAM sections to be tested during an atomic test Set to 1, as minimum (one section per test) If the value is higher than the number of sections in all subsets, all RAM subsets are tested in one pass

表23. STL_SCH_ConfigureRam输出信息

STL_Status_t return value		*pSingleTmStatus output		Returned state
Value	Comments	Value	Comments	
STL_OK	Function successfully executed	STL_NOT_TESTED	-	RAM_CONFIGURED
		STL_ERROR	Possible source of defensive programming error: <ul style="list-style-type: none"> State not allowed Wrong configuration detected STL internal data corrupted 	No state change
STL_KO	Possible source of defensive programming error: <ul style="list-style-type: none"> pSingleTmStatus = NULL pRamConfig = NULL STL internal data corrupted 	Not relevant	Value must not be used	No state change

附加信息：如果返回值设置为STL_KO或*pSingleTmStatus设置为STL_ERROR，则RAM配置不会被应用。

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, leTm状态)

表24. STL_SCH_RunRamTM 输入信息

Allowed states	Parameters	
	Value	Comments
RAM_CONFIGURED	*pSingleTmStatus	See Caution

表25. STL_SCH_RunRamTM输出信息

STL_Status_t return value		*pSingleTmStatus output		Returned state
Value	Comments	Value	Comments	
STL_OK	Function successfully executed	STL_PASSED	-	RAM_CONFIGURED
		STL_PARTIAL_PASSED	-	RAM_CONFIGURED
		STL_FAILED	-	RAM_CONFIGURED
		STL_NOT_TESTED	All subsets are already tested	RAM_CONFIGURED
		STL_ERROR	Possible source of defensive programming error: <ul style="list-style-type: none"> State not allowed Configuration corrupted STL internal data corrupted 	No state change
STL_KO	Possible source of defensive programming error: <ul style="list-style-type: none"> pSingleTmStatus = NULL STL internal data corrupted 	Not relevant	Value must not be used	No state change

7.2.4.4

STL_Status_t STL_SCH_ResetRam

描述: 重置RAM测试。

声明: STL_Status_t STL_SCH_ResetRam(STL_TmStatus_t *pSingleTmStatus)

表26. STL_SCH_ResetRam 输入信息

Allowed states	Parameters	
	Value	Comments
RAM_CONFIGURED	*pSingleTmStatus	See Caution

表27. STL_SCH_ResetRam 输出信息

STL_Status_t return value		*pSingleTmStatus output		Returned state
Value	Comments	Value	Comments	
STL_OK	Function successfully executed	STL_NOT_TESTED	Configuration successfully applied	RAM_CONFIGURED
		STL_ERROR	Possible source of defensive programming error: <ul style="list-style-type: none"> State not allowed Configuration corrupted STL internal data corrupted 	No state change
STL_KO	Possible source of defensive programming error: <ul style="list-style-type: none"> pSingleTmStatus = NULL STL internal data corrupted 	Not relevant	Value must not be used	No state change

附加信息

- 所有子测试完成后，用户需要重置测试模块以再次进行RAM测试。
- 当返回值设置为STL_KO或*pSingleTmStatus被设置为STL_ERROR时，不会执行RAM重置。

7.2.4.5

STL_SCH_DeInitRam

描述：反初始化RAM测试。

声明：STL_Status_t STL_SCH_DeInitRam(STL_TmStatus_t *pSingleTmStatus)

表28. STL_SCH_DeInitRam 输入信息

Allowed states	Parameters	
	Value	Comments
RAM_IDLE RAM_INIT RAM_CONFIGURED	*pSingleTmStatus	See Caution

表29. STL_SCH_DeInitRam 输出信息

STL_Status_t return value		*pSingleTmStatus output		Returned state
Value	Comments	Value	Comments	
STL_OK	Function successfully executed	STL_NOT_TESTED	-	RAM_IDLE
STL_KO	Possible source of defensive programming error: <ul style="list-style-type: none"> pSingleTmStatus = NULL STL internal data corrupted 	Not relevant	Value must not be used	No state change

7.2.5

人工故障的API

7.2.5.1

STL_SCH_StartArtifFailing

描述：设置人工故障配置并启动人工故障功能。

声明：STL_Status_t STL_SCH_StartArtifFailing(const STL_ArtifFailingConfig_t *pArtifFailingConfig)

表30. STL_SCH_StartArtifFailing 输入信息

Allowed states	Parameters	
	Value	Comments
CPU TMx: <ul style="list-style-type: none"> CPU_TMx_CONFIGURED Flash TM: <ul style="list-style-type: none"> FLASH_IDLE FLASH_INIT FLASH_CONFIGURED RAM TM <ul style="list-style-type: none"> RAM_IDLE RAM_INIT RAM_CONFIGURED 	*pArtifFailingConfig	No state change

表 31. STL_SCH_StartArtifFailing 输出信息

STL_Status_t return value	Comments	Output	Comments
STL_OK	Function successfully executed	No output parameter	No state change
STL_KO	Possible source of defensive programming error: <ul style="list-style-type: none"> • pArtifFailingConfig = NULL • configured values are not set for each test module • STL internal data corrupted 		

附加信息：所有后续的 API 调用都会正常执行，除非 STL_Status_t 的返回值被设置为 STL_OK，此时测试模块状态 (*pSingleTmStatus, *pTmListStatus) 被强制设为配置值。

7.2.5.2 STL_SCH_StopArtifFailing

描述：停止人工故障功能。 声明 STL_Status_t STL_SCH_StopArtifFailing(void)

表 32. STL_SCH_StopArtifFailing 输入信息

Allowed states	Parameters	
	Value	Comments
CPU TMx: <ul style="list-style-type: none"> • CPU_TMx_CONFIGURED Flash TM: <ul style="list-style-type: none"> • FLASH_IDLE • FLASH_INIT • FLASH_CONFIGURED RAM TM <ul style="list-style-type: none"> • RAM_IDLE • RAM_INIT • RAM_CONFIGURED 	No input parameter	No state change

表 33. STL_SCH_StopArtifFailing 输出信息

STL_Status_t return value	Comments	Output	Comments
STL_OK	Function successfully executed	No output parameter	No state change
STL_KO	Possible source of defensive programming error: <ul style="list-style-type: none"> • STL internal data corrupted 		

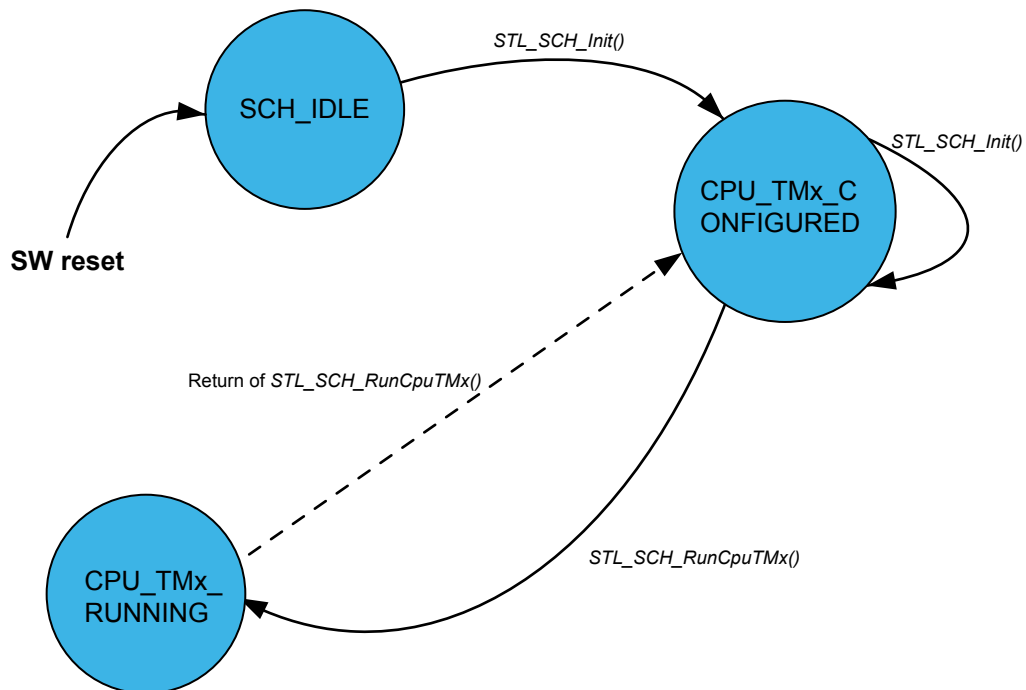
7.3 状态机

每个 CPU 测试模块拥有自己的状态机图，与 CPU 相关联

测试 APIs.

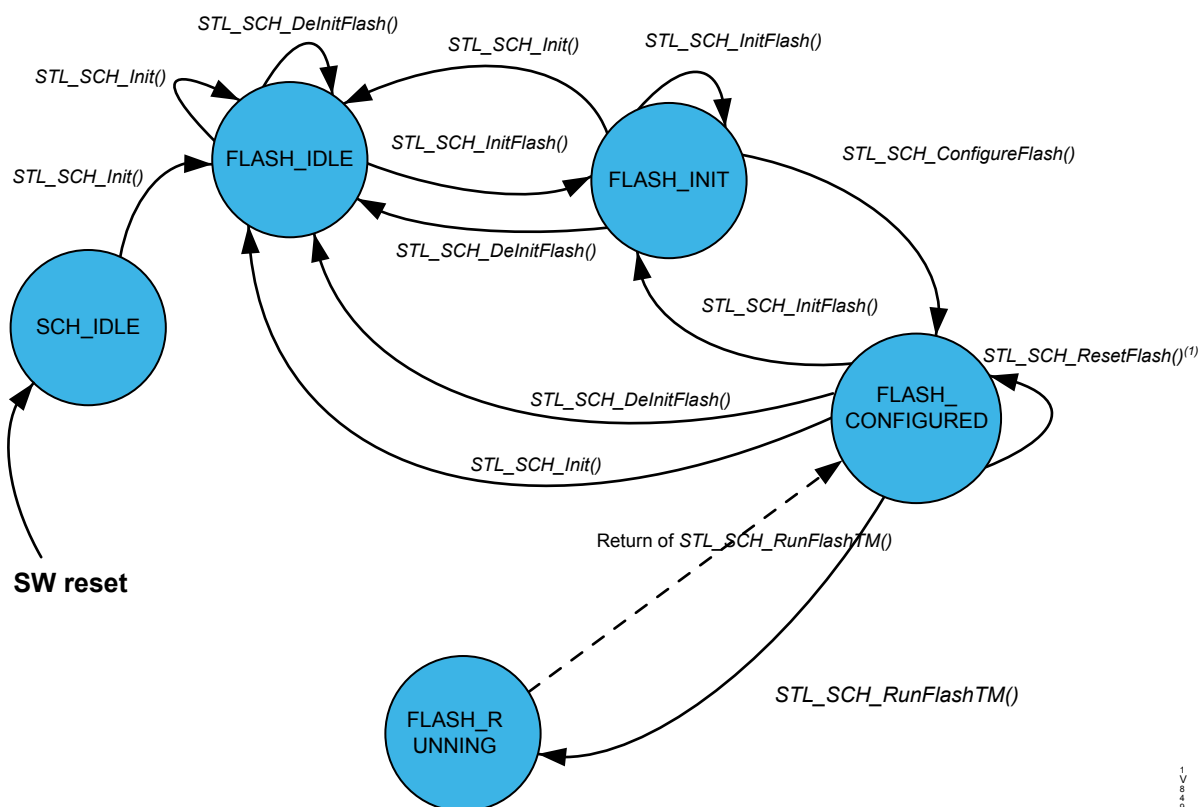
CPU测试 APIs

图11. 状态机图 - CPU 测试 APIs


1
V
7
4
9
6
T
D

Flash memory test APIs

Figure 12. State machine diagram - flash memory test APIs

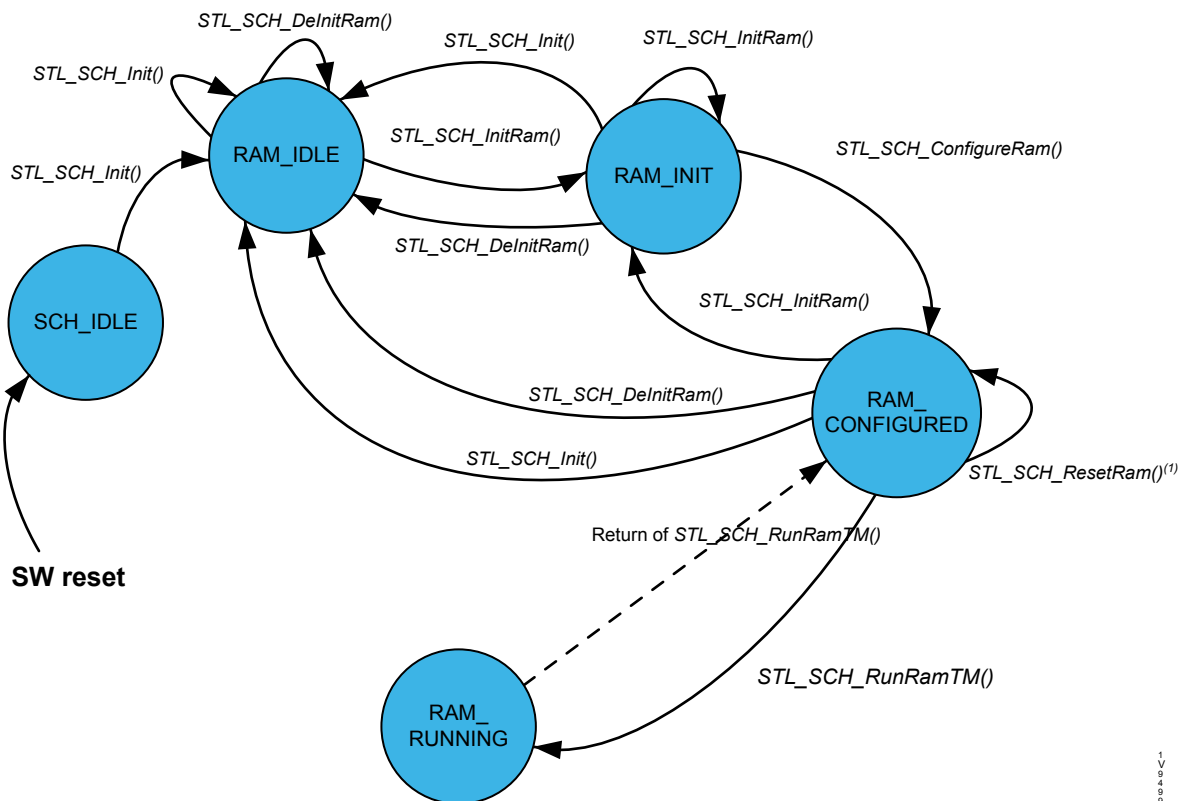


注释（1）：在所有子集测试完成后，用户需要重置闪存测试模块以重新进行测试。

1
V
8
4
9
6
T
D

RAM测试 APIs

图13. 状态机图 - RAM测试 APIs



注释 (1) : 在所有子集测试完成后, 用户需要重置RAM测试模块以再次进行测试。

1
V
9
4
9
9
6
T
D

7.4 API 使用及顺序

用户应用程序必须:

- 在测试期间维护作为参数传递的指针的可用性和完整性。The STL does 复制指针内容, 并直接访问应用程序定义的内存地址。不
- 检查函数返回值的状态 (STL_Status_t), 在检查测试结果 (STL_TmStatus_t 或 STL_TmListStatus_t) 之前。请参见交付的应用程序中的示例。

这些 APIs 相互独立地运行, 因此可以按任意顺序调用。

只有 APIs 用于内存配置和初始化的测试必须在执行这些测试之前被调用。参见第7.3节: 状态机以获取更多细节。

测试流程已简化, 所有测试现在均从C代码中执行。所有模块均为通用模块, 适用于启动和运行时测试。启动和运行时测试之间的区分可通过测试模块的适当序列化和配置实现。在应用复位后, 常规做法是执行包含应用启动前所有内存区域上完整测试集的完整初始化序列。此序列按以下顺序定义:

1. 所有CPU测试
2. 非易失性内存完整性的完整测试
3. 对可用空间的易失性内存进行整体功能测试, 包括专门用于堆栈的区域

Note:

Temporarily suppressing of the memory content backup can be applied to speed up initial testing of huge RAM areas where user does not need to preserve the memory content during this test. For more details see Section 4.3.7: RAM backup buffer. Functional test is not executed over areas containing program code and data when the code is executed from RAM.

4. 特定客户测试

之后，在运行时，可以更改测试的顺序，并以更宽松的方式执行。被测试的内存区域可以减少。测试过程甚至可以动态地修改，优先关注那些存储最近执行的安全相关代码和数据的区域。这在考虑以下因素时尤为重要：

- 可用的应用过程安全时间
- 系统整体性能
- 申请的具体状态

7.5 用户参数

除了在 *API* 内直接设置的参数外，在 `stl_user_param_template.c` 文件中还有一些需要自定义的参数。它们位于代码中，并带有以下注释：

```
/* customisable */
```

摘自 `stl_user_param_template.c`：

```
/* 闪存配置 */ #define STL_ROM_START (0x08000000U) /* 可定制的 */ #define  
fine STL_ROM_END (0x0801FFFFU) /* 可定制的 */
```

的 定制取决于STM32产品以及其使用 `r` 选择。

```
/* TM RAM备份缓冲区配置 */ ... /* 用户应将缓冲区定位在RAM中 */ /* RAM备份缓冲区  
放置在"backup_buffer_section"中。 */ /* "backup_buffer_section"段在散列文件中定义 */
```

自定义取决于用户的选择。

剩余的用户参数由标志定义，可以查看以下文件：

- `stl_user_param_template.c`: 是否使用RAM备份缓冲区
- `stl_util.c`: 使用软件或硬件 *CRC* 计算
- `stl_stm32_hw_config.h`: 如果使用*CRC*硬件，请根据STM32设备选择正确的*CRC* IP配置

Refer to Section 5.5.2: Steps to build an application from scratch for the flag configuration check.

八 测试示例

图14展示了通过STL调度器进行的可能的STL API调用序列以及由STL提供的返回信息（参见图1和表2）。

Figure 14. Test flow example

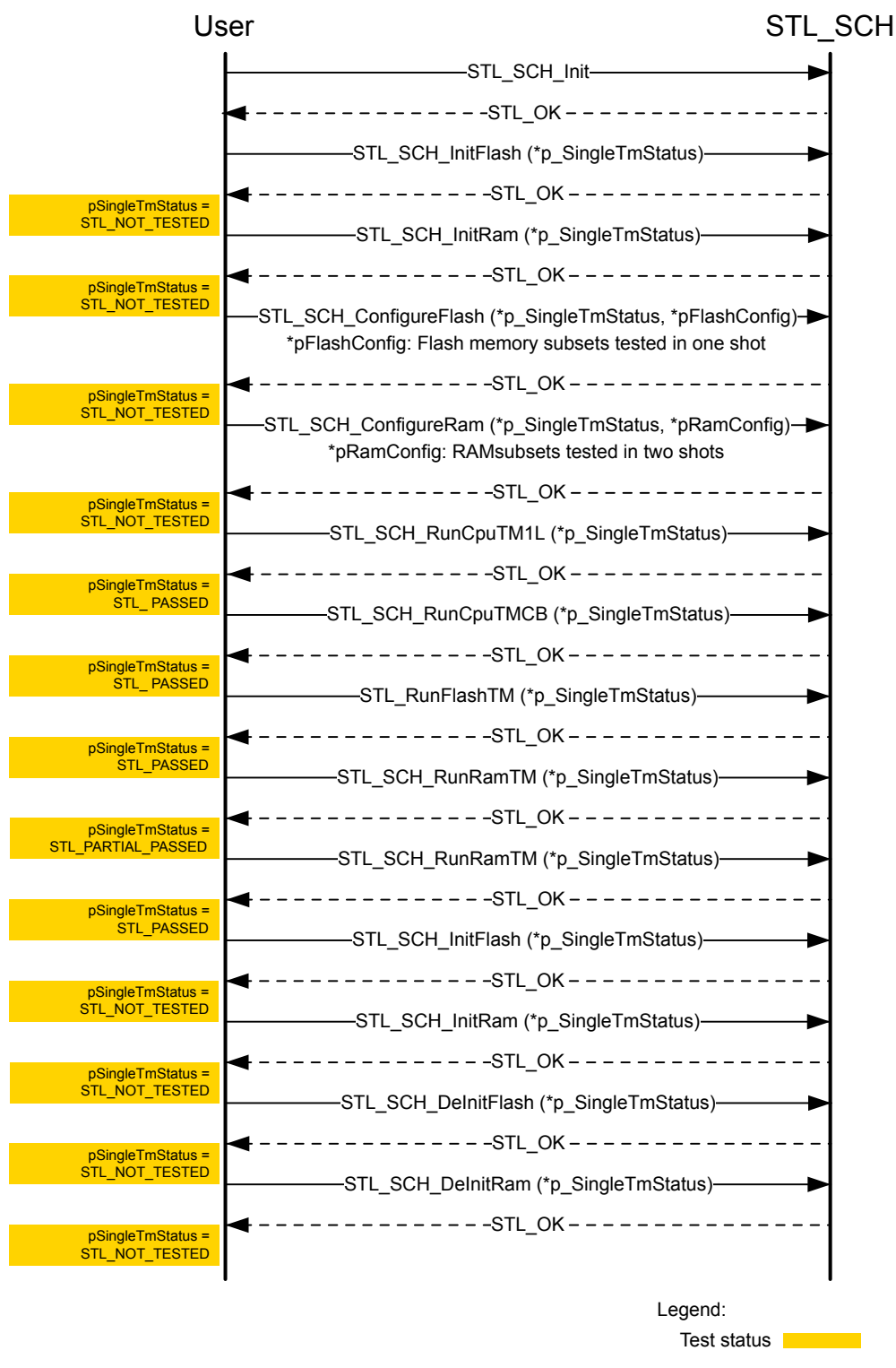


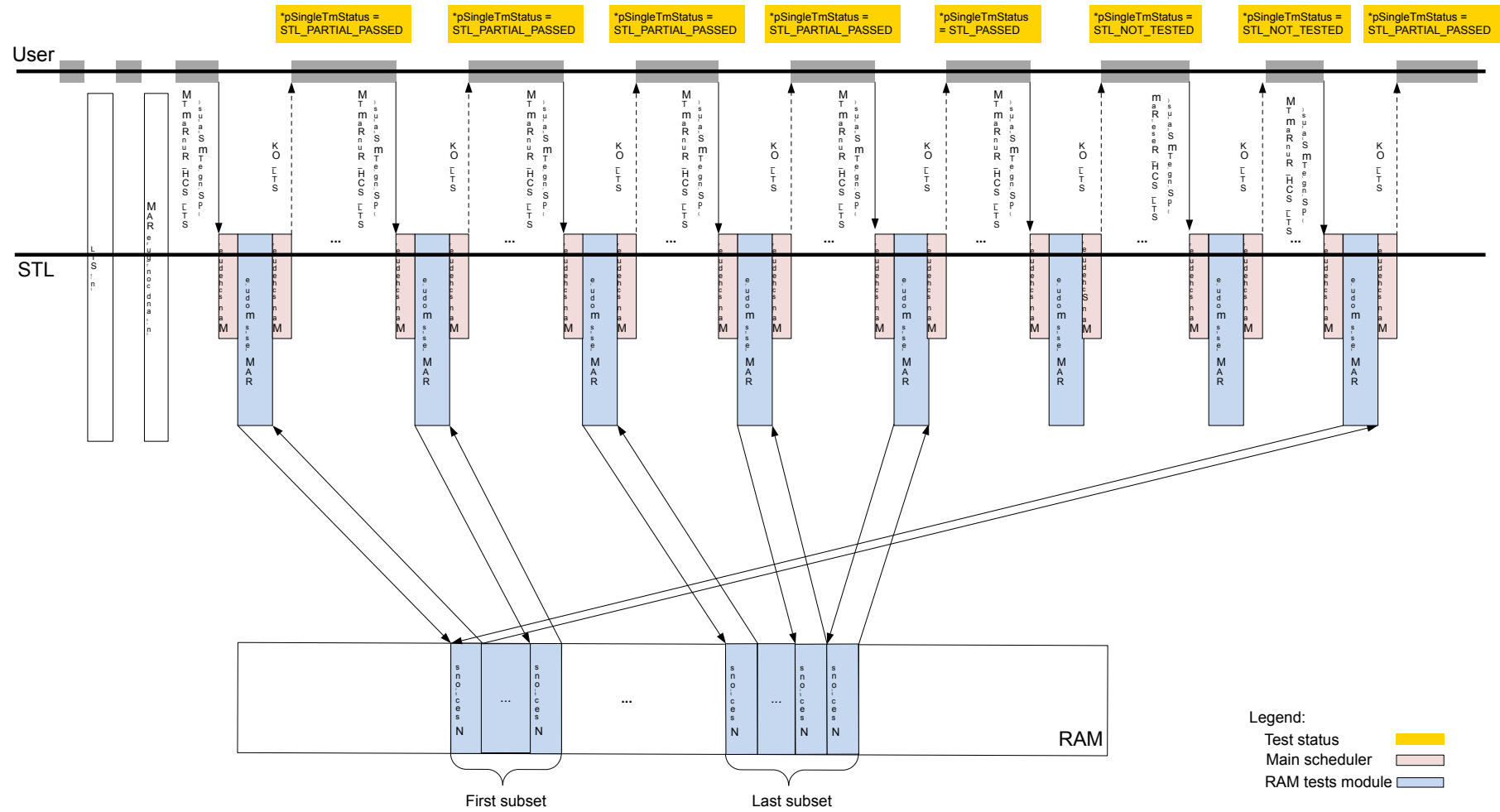
图15展示了一个详细的闪存测试流程处理示例：

- 两个闪存子集的应用
- 函数的使用 – STL_SCH_RunFlashTM → 仅执行闪存内存测试模块 – STL_SCH_ResetFlash
- 函数返回值
- 闪存测试模块结果值： pSingleTmStatus →，在此情况下，它包含闪存测试的结果

图16展示了一个详细的RAM测试流程处理示例： {v*}

- 使用两个RAM子集
- 函数的使用： – STL_SCH_RunRAMTM → 仅执行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内存测试的结果

Figure 16. RAM test flow example



九 STL: 执行时间详情

下表中的数据是通过如第4.2节所述的测试设置获得的：STL性能数据

表34. 集成测试

Test	Duration (in μ s)		Tested memory
	Hardware CRC	Software CRC	
STL_SCH_InitFlash	5	5	-
STL_SCH_ConfigureFlash	8	8	-
STL_SCH_RunFlashTM	582	3183	17408 bytes tested
STL_SCH_InitRam	5	5	-
STL_SCH_ConfigureRam	7	8	-
STL_SCH_RunRamTM	137484	137484	163807 bytes tested
STL_SCH_RunCpuTM1L	42	24	-
STL_SCH_RunCpuTM7	18	18	-
STL_SCH_RunCpuTMCB	7	7	-

十 应用专用测试未包含在ST固件自检库中

用户必须关注所有剩余的必要测试，涵盖应用特定的 *MCU* 部分，这些部分未包含在 ST 固件库中：

- 模拟部分测试 (ADC/DAC, 多路复用器)
- 数字I/O测试
- 外部寻址
- 外部沟通
- 定时和中断
- 系统时钟频率测量。

Note: *The clock frequency measurement is not an integrated part of the STL package. The clock testing module is provided as full source format within STL integration example to demonstrate the capability of implementing additional user defined testing modules which can be included at the STL flow. For more details refer to Section 10.5: Extension capabilities STL library.*

这些组件的有效解决方案高度依赖于应用场景和设备外围能力。应用必须尽可能精确地遵循其设计最初阶段建议的测试原则。

这种方法经常导致硬件和软件层面的冗余。

硬件方法可以基于：

- 输入和/或输出的乘法
- 参考点测量
- 环回读取控制在模拟或数字输出（如 *DAC*, *PWM*, *GPIO*）处
- 配置保护。

软件方法可以基于：

- 时间上的重复，多次获取，多次检查，决策，或在不同时间进行的计算 {v*} 或通过不同的方法
- 数据冗余（数据副本、奇偶校验、纠错码/检错码、校验和、协议）
- 合理性检查（有效范围、有效组合、预期变化或趋势）
- 周期性与发生检查（流程及时间控制中的发生）
- 定期检查正确配置（例如，读取配置寄存器）。

10.1 模拟信号

测量值必须进行一致性检查，并通过其他冗余通道的测量进行验证。空闲通道可用于通过测试模拟多路复用器（用于应用中的）来读取某些参考电压。内部参考电压也必须进行检查。

一些STM32微控制器设备具有两个（甚至三个）独立的 *ADC* 块。为确保结果的可靠性，出于安全考虑，使用两个不同的 *ADC* 块对同一通道进行多次转换。结果可以通过以下任一种方式获得：

- 从单一渠道进行多次收购
- 比较冗余通道后进行平均操作。以下是一些提示，用于测试STM32微控制器设备上的模拟部分功能。

ADC输入引脚断开

ADC输入引脚的断开可以通过在被测试的引脚上施加额外的信号源进行测试。

- 一些STM32微控制器设备在模拟输入上具有内部下拉或上拉电阻激活功能。它们还可以具有一个自由引脚，其功能为DAC或数字GPIO输出。这些引脚中的任何一个都可以用作ADC的已知参考输入。
- 某些STM32微控制器设备具有路由接口。该接口可用于引脚之间的内部连接，以实现：– 测试环回 – 额外的信号注入 – 在另一个独立通道上的重复测量。

Note: *The user must prevent any critical voltage injection into an analog pin. This can happen when digital and analog signals are combined and different power levels are applied to analog and digital parts ($V_{DD} > V_{DDA}$).*

内部参考电压和温度传感器（VBAT 用于某些设备）

- 这些信号之间的比率可以在允许的范围内验证。
- 可以进行额外的测试，当{v*}电压已知时。

ADC钟表

测量 ADC 转换时间（通过定时器）可以用来测试独立的 ADC 时钟功能。

DAC输出功能

免费 ADC 通道可以用来检查 DAC 输出通道是否正常工作。

路由接口可以在连接 ADC 输入通道和 DAC 输出通道时使用。

比较器功能

比较已知电压与 DAC 输出或内部参考电压可用于在另一个比较器输入上测试比较器输出。通过在测试引脚上启用下拉或上拉，并将此信号与 DAC 电压作为参考进行比较，可以测试模拟信号断开。

运算放大器

功能可以通过将已知的模拟信号施加（或测量）到运算放大器(OPAMP)的输入引脚，并通过ADC内部测量输出电压来测试。OPAMP的输入信号也可以通过ADC（在另一通道上）进行测量。

10.2 数字输入/输出

B类测试也必须检测数字I/O的任何故障。这可以通过合理性检查与其他一些应用部分共同实现。例如，当加热/冷却数字控制开启/关闭时，必须检查来自温度传感器的模拟信号的变化。通过向GPIOx_LCKR寄存器中的锁定应用正确的锁定序列，可以锁定选定的端口位。此操作可防止端口配置的意外更改。在这种情况下，重新配置只能在下一个复位序列时进行。此外，位带功能可用于对SRAM和外设寄存器进行原子操作。

10.3 中断

事件发生的时间和周期性必须进行检查。可以采用不同的方法；其中一种方法使用一组增量计数器，其中每个中断事件都会增加特定的计数器。计数器中的值随后会定期与其他独立的时间基准进行交叉检查。上一周期内发生的事件数量取决于应用需求。

配置锁定功能可用于通过 TIMx_BDTR 寄存器控制的 {v*} 级别来保护定时器寄存器设置。未使用的中断向量必须重定向到通用错误处理程序。如果可能，对于非安全相关的任务，轮询更可取，以简化应用程序中断方案。

10.4 沟通

在通信会话期间进行的数据交换必须在数据包中包含冗余信息时进行检查。奇偶校验、同步信号、CRC校验和、分块重复或协议编号可用于此目的。如果需要，健壮的应用软件协议栈（如TCP/IP）可以提供更高层次的保护。通信事件的时间周期性和发生时间以及协议错误信号必须持续不断地进行检查。

用户可以在专用的安全手册中找到更多信息和方法。

10.5 扩展功能 STL库

此框架版本相较于此 **STL** 库的先前版本，具有更简单且更灵活的实现方式（参见第1.2节：参考文档），这使得扩展更加容易。即使采用了新的应用格式，该框架仍保留相同的自测方法集以符合IEC 60730标准，这些方法已由库的先前版本实现：

- 寄存器测试在 *CPU TMs*
- 32位 *CRC* 计算 兼容 STM32 HW *CRC* 单元，位于 Flash *TM*
- March C测试遵循RAM的物理地址顺序 *TM*
- 由LSI触发的定时器，用于检查时钟 *TM*的频率定义 ned 在 *STL* 集成示例

新框架版本的主要改进包括：

- 模块化
- 支持部分测试
- 基于配置和参数化结构
- 启动和运行时测试模块之间没有区分
- *CRC*基于STM32CubeProgrammer命令行功能提供的格式的计算支持
- 预编译和固定的关键通用模块的目标代码格式
- 通用模块的执行不依赖于驱动程序或编译器
- 错误处理包括防御性编程结果的报告
- 人工故障控制功能，用于验证模块的正确集成，无需额外的测试代码 {v*}
- 通过附加专用模块实现易于扩展。

附加的特定测试模块实现示例可在固件包集成示例中找到。一个基于两个独立时钟源交叉检查测量方法的特定测试模块以开源格式与固件包集成示例一起提供。该模块必须由终端用户进行调整，以考虑所应用时钟系统的特定配置依赖。

本模块采用与库之前版本中已应用的相同测量原理。用于频率比较的硬件必须首先进行配置（TIM16的通道1由LSI触发），以便在调用相关的API之前启动时钟测量。此硬件配置在main.c文件的STL_Init()过程结束时进行。API被编写为与集成在STL中的常规API兼容，因此在它的声明中应用了相同的格式：

```
STL_Status_t STL_SCH_运行时钟测试(STL_TmStatus_t *pSingleTmStatus)
```

传递的参数充当指向时钟模块测量状态的指针，该函数本身提供STL_KO与STL_OK的返回状态，并在防御性编程失败时执行常规的STL模块。如果时钟测量硬件处于活动状态，并且由上一次测量周期更新的新周期值（设置为8个连续的LSI周期）在预期的时间间隔（由宏CLK_LimitLow和CLK_LimitHigh定义）被检测到，则模块测量状态值被更改为STL_PASSED。如果未检测到，则按照常规的API模块设置为STL_FAILED。这也是当模块被人工强制失败时的情况。

以类似的方式，用户可以集成以下模块。例如，任何栈加固技术，如栈边界区域检查或实现看门狗测试和维护，不再默认包含在此新包中。这些测试的源代码可在本库的旧版本中找到 see [2]。如需了解普遍认可的安全方法，这些方法并非家庭标准特别要求，请参阅[1]。它们可能有助于提高用户应用程序的健壮性。

11 符合IEC、UL和CSA标准

关键的IEC标准是IEC 60730-1和IEC 60335-1，自第四版起与UL/CSA 60730-1和UL/CSA 60335-1协调。之前的UL/CSA版本则另外引用UL1998标准。

这些标准定期更新。标准中收集的所有法规范围非常广泛；涉及微控制器通用部件软件自检要求的章节非常具体。在大多数情况下，提供的更新不会影响这些标准中的任何具体部分。因此，过时的认证仍可符合并保持有效于标准的新版本。

所需的有关详细条件见下文：{v*}

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

由IEC 60730-1:2010 H.2.22定义的三类是：

- 类别A：不预期被依赖于应用安全性的控制功能。
- B类：控制功能是指旨在防止被控设备处于不安全状态的控制功能。控制功能的失效不会直接导致危险情况。
- 类C：旨在防止如爆炸等特殊危险，或其故障可能直接导致器具危险的控制功能。

对于应用安全保护功能的可编程电子组件，IEC 60335-1标准要求纳入软件措施以控制表R.1和R.2中规定的故障和错误情况：

- 表 R.1 总结了与Class B等级要求相当的一般条件
- 表R.2总结了与Class C级别所规定的具体条件相当的条件。

B级软件的要求，即本用户手册的主题，是为了防止在设备其他部分发生其他故障时出现危险而定义的。在这种情况下，设备在故障后运行自检软件。在执行安全关键程序期间发生的意外软件故障，由于在该层级所需的另一应用的冗余软件程序或硬件保护功能，不一定导致危险。

在C级计数中，无需硬件保护以防止安全关键软件中的任何故障导致潜在危害。要符合这一等级，所需的测试比通常适用于标准工业微控制器（如STM32）的测试更为严格。可接受的解决方案通常需要在系统层面实现特定的硬件冗余，例如双通道结构。

如需了解更严格的测试方法，请参阅工业文档 [1]。

IEC 60730-1 定义了适用于设计B类控制功能的可接受架构集：

- 单通道功能测试。单个CPU按需执行软件控制功能。软件启动时执行功能测试。这确保了所有关键功能正常工作。
- 单通道，具有周期性自检功能。单个CPU执行软件控制功能。嵌入式周期性测试检查系统的各种关键功能，而不影响计划控制任务的性能。
- 双通道（同构或异构）并进行比较。软件设计用于在两个独立的CPU上执行控制功能（相同或不同）。两个CPU比较内部信号用于故障 {v*} 在执行任何安全关键任务时的检测

Note: *This structure is recognized to comply with Class C level also. A common principle is that whatever method complies with Class C automatically complies with Class B.*

以下表格列出了STL所采用的方法及其对标准的引用。STL专注于微控制器中在所有应用中复用的通用组件。其他部分的测试由终端用户负责，因为它们的测试大多是特定于应用的，并且可以在系统设计的规划阶段有效完成。参见第10节：未包含在ST固件自检库中的应用特定测试，以获取有关如何处理这些应用特定测试的更多信息。

表 35. 由X-CUBE-CLASSB库覆盖的IEC 60335-1组件，通过IEC-60730认可的方法

-1

Component of Table R.1 (IEC 60335-1: Annex R)		Class B	References to IEC 60730-1: Annex H)	Fault/error	Safety method applied at X-CUBE-CLASSB	Note
1. CPU	1.1 CPU registers	X	H.2.16.5 H.2.16.6 H.2.19.6	Stuck at	Periodic run of the <i>STL</i> TM1L, TM7, and TMCB CPU test modules	Combination of functional and pattern tests of the <i>CPU</i> registers,(general-purpose R0-R12, special-purpose main and process stack pointers R13, program status APSR and CONTROL registers) ⁽¹⁾
	1.2 Instruction decoding and execution		N/A			Not required for Class B
	1.3 Program counter	X	H.2.18.10.2 H.2.18.10.4	Stuck at	N/A End-user responsibility	Logical and time slot program sequence monitoring, implementation of watchdogs
	1.4 Addressing		N/A			Not required for Class B
	1.5 Data path instruction decoding		N/A			Not required for Class B
2. Interrupt handling and execution		X	H.2.18.10.4 H2.18.18	No interrupt or too frequent interrupts	Handshake of results is applied at the interrupt associated with a clock cross-check measurement module	End-user responsibility for the other interrupts implemented at application
3. Clock		X	H.2.18.10.1 H.2.18.10.4	Wrong frequency	Periodic run of clock cross-check module. Added at open source format as a user specific test module within the firmware integration example	Clock cross-check measurement done between two independent clock sources (system clock and LSI)
4. Memory	4.1 Invariable memory	X	H.2.19.3.1 H.2.19.3.2 H.2.19.8.2	All single bit faults	Periodic execution of the <i>STL</i> FlashTM test module	ECC enable under end-user responsibility ⁽²⁾
	4.2. Variable memory	X	H.2.19.6 H.2.19.8.2	DC fault	Periodic execution of the <i>STL</i> RamTM test module	ECC or parity enable under end-user responsibility ⁽²⁾
	4.3 Addressing (relevant for variable and invariable memory)	X	H.2.19.8.2	Stuck at	-	Tested indirectly by execution of the applied memory test modules ECC enable under end-user responsibility ⁽²⁾
5. Internal data path	5.1 Data	X	H.2.19.8.2	Stuck at	-	
	5.2 Addressing	X	H.2.19.8.2	Wrong address	-	
6. External communication		X	-	-	N/A End-user responsibility	-
7. I/O periphery		X	-	-	N/A End-user responsibility	-
8. Monitoring devices and comparators			N/A			Not required for Class B
9. Custom chips		X	-	-	N/A	-

1. CPU registers R14 (LR) and R15 (PC) are tested indirectly via defensive programming methods.

2. For availability and functionality of concrete embedded hardware safety feature, refer to the product user and safety manual.

修订历史 {v*}

表 36. 文档修订历史

Date	Version	Changes
03-Feb-2025	1	Initial release.

术语表

ADC 模数转换器

SDK 软件开发工具包

AEABI ARM® 嵌入式应用二进制接口

STL 自检库

API 应用程序编程接口

测试模块 TM

APSR CPU状态寄存器

板支持包

B类

中等层级的法规，针对家用电器的安全 (UL/CSA/IEC 60730-1/60335-1)

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

FPU 浮点运算单元 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 低层

MCU 微控制器单元 MPU 内存保护单元 MSP 主栈指针 OPA MP 运算放大器 PSP 进程栈指针 PWM 脉宽调制 RAM 随机存取存储器

目录

1	一般信息	2
1.1	目的和范围	2
1.2	参考文件	2
2	STM32Cube 概述	3
2.1	什么是 STM32Cube?	3
2.2	这款软件如何补充 STM32Cube?	3
3	STL概述	4
3.1	架构概述	4
3.2	支持的产品	5
四	STL描述	6
4.1	STL功能描述	6
4.1.1	调度器原理	6
4.1.2	CPU Arm®核心测试	7
4.1.3	Flash存储器测试	8
4.1.4	RAM测试	10
4.2	STL性能数据	12
4.2.1	STL执行时间	12
4.2.2	STL代码和数据大小	13
4.2.3	STL堆使用情况	13
4.2.4	STL堆使用情况	13
4.2.5	STL中断屏蔽时间	13
4.3	STL用户约束	13
4.3.1	特权级	14
4.3.2	RCC资源	14
4.3.3	CRC资源	14
4.3.4	中断管理	14
4.3.5	DMA	15
4.3.6	支持的存储器	15
4.3.7	RAM备份缓冲区	15
4.3.8	内存映射	15
4.3.9	处理器模式	16
4.4	最终用户集成测试	16
4.4.1	测试1: 正确的STL执行	16
4.4.2	测试2: 正确的STL错误信息处理	16
五	包描述	17

5.1 一般描述	17
5.2 架构	17
5.2.1 STM32Cube HAL	17
5.2.2 板支持包 (BSP)	18
5.2.3 STL	18
5.2.4 用户应用程序示例	18
5.2.5 STL完整性	18
5.3 文件夹结构	19
5.4 API	19
5.4.1 合规性	19
5.4.2 依赖性	20
5.4.3 详情	20
5.5 应用: 编译过程	20
5.5.1 构建交付的STL示例的步骤	20
5.5.2 从头开始构建应用程序的步骤	21
6 硬件和软件环境设置	23
6.1 硬件设置	23
6.2 软件设置	23
6.2.1 开发工具链和编译器	23
6.2.2 CRC 工具设置	23
7 STL: 用户API和状态机	24
7.1 用户结构	25
7.2 用户API	25
7.2.1 通用API	26
7.2.2 CPU Arm®核心测试API	26
7.2.3 闪存内存测试API	26
7.2.4 RAM测试API	27
7.2.5 人工故障API	31
7.3 状态机	35
7.4 API使用 and 序列化	36
7.5 用户参数	39
8 测试示例	41
9 STL: 执行时间细节	45
10 未包含在ST固件自检库中的应用特定测试	46
10.1 模拟信号	46
10.2 数字输入/输出	47



10.3 中断	47
0.4 通信	48
10.5 扩展功能	48
STL库	48
11 符合IEC、UL和CSA	49
标准	49
修订历史	51
术语表	51
表格列表	52
	56

图目录。

表格列表

表 1. 适用产品	1	表 2. STL 返回信息	7
表 3. STL 执行时间, 时钟为 84 MHz	13	表 4. STL 代码大小和数据大小 (以字节为单位)	13
表 5. STL 最大中断屏蔽信息	13	表 6. STL_SCH_Init 输入信息	26
表 7. STL_SCH_Init 输出信息	26	表 8. STL_SCH_RunCpuTMx 输入信息	27
表 9. STL_SCH_RunCpuTMx 输出信息	27	表 10. STL_SCH_InitFlash 输入信息	27
表 11. STL_SCH_InitFlash 输出信息	27	表 12. STL_SCH_ConfigureFlash 输入信息	28
表 13. STL_SCH_ConfigureFlash 输出信息	29	表 14. STL_SCH_RunFlashTM 输入信息	29
表 15. STL_SCH_RunFlashTM 输出信息	30	表 16. STL_SCH_ResetFlash 输入信息	30
表 17. STL_SCH_ResetFlash 输出信息	30	表 18. STL_SCH_DeInitFlash 输入信息	31
表 19. STL_SCH_DeInitFlash 输出信息	31	表 20. STL_SCH_InitRam 输入信息	31
表 21. STL_SCH_InitRam 输出信息	31	表 22. STL_SCH_ConfigureRam 输入信息	32
表 23. STL_SCH_ConfigureRam 输出信息	33	表 24. STL_SCH_RunRamTM 输入信息	33
表 25. STL_SCH_RunRamTM 输出信息	34	表 26. STL_SCH_ResetRam 输入信息	34
表 27. STL_SCH_ResetRam 输出信息	34	表 28. STL_SCH_DeInitRam 输入信息	35
表 29. STL_SCH_DeInitRam 输出信息	35	表 30. STL_SCH_StartArtifFailing 输入信息	36
表 31. STL_SCH_StartArtifFailing 输出信息	36	表 32. STL_SCH_StopArtifFailing 输入信息	36
表 33. STL_SCH_StopArtifFailing 输出信息	36	表 34. 集成测试	45
表 35. IEC 60335-1 组件由 X-CUBE-CLASSB 库通过 IEC-60730-1 承认的方法覆盖	50	表 36. 文档修订历史	51

图录

图1. STL架构	4	图2. 单个测试控制调用架构	7
图3. Flash内存测试: CRC原理	9	图4. Flash内存测试: CRC用例与程序区域	10
图5. RAM测试: 使用	12	图6. 软件架构概览	17
图7. 项目文件结构	19	图8. IAR™构建后操作截图	22
图9. CRC工具命令行	23	图10. STM32 Nucleo开发板示例	37
图11. 状态机图 - CPU测试 API示例	38	图12. 状态机图 - Flash内存测试 API示例	39
图13. 状态机图 - RAM测试 API示例	41	图14. 测试流程示例	43
图15. Flash内存测试流程示例	44	图16. RAM测试流程示例	44

重要通知 – 请仔细阅读

STMicroelectronics International NV及其附属公司（“ST”）保留随时且无需通知地对ST产品和/或本文件进行修改、更正、改进、变更和优化的权利。

本文件仅用于获取与ST产品相关的一般信息。因此，您在此同意仅将本文件用于获取与ST产品相关的一般信息。您进一步承认并同意，不得将本文件用于或涉及任何法院、仲裁、机构、委员会或其他法庭的法律或行政程序，也不得用于或涉及任何类型的行动、诉因、诉讼、索赔、指控、要求或争议。您进一步承认并同意，本文件不得被解释为任何形式的承认、认可或证据，包括但不限于ST或其关联公司就任何责任、过失或义务的承认，或有关本文件中所含信息的准确性或有效性，或涉及任何声称的产品问题、故障或缺陷。ST不保证本文件的准确性或无错误，并明确否认所有关于本文件中所含信息准确性的明示或暗示担保。因此，您同意在任何情况下，ST或其关联公司均不对您因依赖或使用本文件而产生的任何直接、间接、后果性、示范性、偶然性、惩罚性或其他损害，包括利润损失，承担任何责任。

P 购买者应在下单前获取ST产品的最新相关信息。ST产品根据ST的条款和条件进行销售。
s 所有在订单确认时已生效的条款，包括但不限于其中的保修条款。

s 的

请注意，ST产品未设计用于上述条款中描述的一些特定应用或环境，并且 {v*}
c 条件

购买者对其选择、挑选和使用ST产品的行为完全负责，ST不对其应用支持或购买者产品的设计承担任何责任。

信息提供者认为所提供的信息是准确且可靠的。然而，ST不对使用此类信息所产生的后果承担任何责任，也不
于任何侵权行为，包括第三方的专利或其他权利可能由此产生。未经许可，不得使用任何知识产权。
g 由ST在此处授予的

对

若ST产品被转售且其条款与本文件所述信息不同，则使ST授予的任何保修失效。

ST 安 ST标志是ST的商标。如需了解有关ST商标的更多信息，请访问www.st.com/trademarks。所有其他产品或服务名称
是的 各自所有者的财产

本文件中的信息取代并替换所有先前版本中的信息。

© 2025 STMicroelectronics – All rights reserved