
**STM32应用中UL/CSA/IEC 60730-1/60335-1的
B类认证获取指南**

前言

安全在电子应用领域起着越来越重要的作用。在电子设计中，组件的安全要求水平不断上升，电子设备制造商将很多新技术解决方案纳入了新组件设计。用于提高安全的软件技术不断涌现。硬件和软件的安全要求相关标准也在持续开发中。

由IEC（国际电工委员会）、UL（美国保险商实验室）及CSA（加拿大标准协会）颁布的全球公认标准指定了当前安全建议和要求，并被TUV及VDE（大部分在欧洲）、UL及CSA（目标市场在美国和加拿大）等机构归入依从性、验证及认证程序。

对于基于STM32 32位ARM® Cortex® 微控制器（受这些要求和认证管制）的应用，该应用笔记和相关软件X-CUBE-CLASSB的主要目的是促进和加速用户软件开发及认证程序。

安全软件包（自检库-STL）收集通用的测试组，这些测试组主要用于STM32微控制器的通用逻辑块。STL组基于独特的STM32Cube接口，带有特定的HAL（硬件抽象层）服务和ST发布的专用STM32产品驱动。系列差异被产品特定测试和新增设置所覆盖（如CPU内核、RAM设计、时钟控制）。

用户可将STL软件包和专用HAL驱动连同附加的产品特定测试及设置一起纳入最终用户项目。STL软件包的实现示例可用于主流STM32F0和STM32F3、高性能STM32F2和STM32F4及低功耗STM32L0和STM32L1系列的特定产品。每个例子都包括三个项目（IAR™-EWARM、Keil® MDK-ARM® 及Ac6 Eclipse™ 环境和工具链），建立在专用ST评估板上。

由于HAL服务统一的Cube接口，STL包的公用部分可重新用于STM32系列的任何其他微控制器。

用户必须了解STL软件包的方法及所用技术已预先认证。提供的示例说明如何将STL软件包和相关FW（HAL驱动）集成到应用中，最终实现和功能始终应由认证机构从应用层面进行认证。

注：意法半导体正逐步开发可支持新产品的衍生固件。为获得支持和可用示例的最新相关信息，请联系您的本地ST销售办事处。

目录

1	参考文档	6
2	软件包变化概述	7
3	STL软件包之间的主要差别（从产品的角度来看）	10
3.1	CPU测试	12
3.2	时钟测试和时基间隔测量	12
3.3	SRAM测试	12
3.4	Flash存储器完整性测试	14
3.5	启动和系统初始化	15
3.6	固件配置参数	15
3.7	固件集成	18
3.8	HAL驱动接口	18
4	符合IEC，UL和CSA标准	20
4.1	STL固件包所含的通用测试	21
4.2	应用相关特定测试不在ST固件自检库范围内	23
4.2.1	模拟信号	23
4.2.2	数字I/O	24
4.2.3	中断	24
4.2.4	通信	24
4.3	安全生命周期	24
5	B类软件包	27
5.1	使用的通用软件原理	27
5.1.1	故障安全模式	27
5.1.2	安全相关变量和栈边界控制	27
5.1.3	流程控制程序	29
5.2	库的工具特定集成	30
5.2.1	软件包所含项目	30
5.2.2	启动文件	30
5.2.3	定义新的安全变量和受检内存区	31
5.2.4	应用实现示例	32

5.3	执行定时测量和控制	32
5.4	软件包配置和调试	36
5.4.1	配置控制	36
5.4.2	冗长诊断模式	37
5.4.3	软件包调试	38
6	B类安全方案结构	40
6.1	与用户应用的集成	40
6.2	启动自检说明	42
6.2.1	CPU启动自检	43
6.2.2	看门狗启动自检	43
6.2.3	Flash存储器完整校验和自检	44
6.2.4	完整RAM March-C自检	45
6.2.5	时钟启动自检	46
6.2.6	控制流程检查	47
6.3	定期运行时自检初始化	47
6.4	运行时定期自检说明	48
6.4.1	运行时自检结构	48
6.4.2	简化版CPU运行时自检	49
6.4.3	栈边界运行时测试	50
6.4.4	时钟运行时自检	50
6.4.5	局部Flash CRC运行时自检	51
6.4.6	运行时测试中的看门狗服务	52
6.4.7	部分RAM运行时自检	52
7	版本历史	56

表格索引

表1.	STL软件包概述	7
表2.	FW结构的主要组织	7
表3.	通用STL软件包的结构	8
表4.	产品特定STL软件包结构	8
表5.	集成支持文件	9
表6.	不同STM32微控制器之间的兼容性	11
表7.	如何管理兼容性及配置STL软件包	16
表8.	所有STL栈程序使用的HAL驱动概述	18
表9.	必须在B类标准依从性下测试的MCU部件	21
表10.	相关ST软件包MCU相关测试中采用的方法	22
表11.	用于定时测量的信号	35
表12.	结果对比	36
表13.	STL进程可能与用户SW发生的冲突	41
表14.	RAM物理地址顺序被组织成16字块	45
表15.	RAM部分测试的March C阶段	55
表16.	版本历史	56
表17.	中文版本历史	57

图片目录

图1.	RAM存储器配置示例.....	28
图2.	控制流程四步检查原理.....	29
图3.	诊断LED定时信号原理.....	33
图4.	启动期间典型的测试时序.....	34
图5.	运行期间典型的测试时序.....	35
图6.	冗长模式下STM32 demo超级终端输出窗口.....	38
图7.	启动和定期运行时自检集成入应用.....	40
图8.	启动自检结构.....	42
图9.	CPU启动自检结构.....	43
图10.	看门狗启动自检结构.....	44
图11.	Flash启动自检结构.....	45
图12.	RAM启动自检结构.....	46
图13.	时钟启动自检子结构.....	47
图14.	运行时定期自检初始化结构.....	48
图15.	定期运行时自检和时基中断服务结构.....	49
图16.	CPU灯运行时自检结构.....	49
图17.	栈上溢运行时测试结构.....	50
图18.	时钟运行时自检结构.....	51
图19.	局部Flash CRC运行时自检结构.....	51
图20.	部分RAM运行时自检结构.....	53
图21.	部分RAM运行时自检 - 故障耦合原理（无加扰）.....	54
图22.	部分RAM运行时自检 - 故障耦合原理（有加扰）.....	54

1 参考文档

在应用或修改STL栈时，或在开发新栈时，ST提供了多个文件供参考，或根据需求提供完整的测试报告。

ST提供了安全手册来描述如何在STM32产品对其他的安全标准进行实现，部分安全手册已经可以下载。还有一部分在准备中。为控制外设或确保针对噪声发射及噪声灵敏度的系统电磁兼容性（EMC），www.st.com提供有描述具体方法的应用笔记。

关于错误处理技术的更多信息，请参考以下应用笔记：

- AN4750 STM32应用内的软件错误处理。

关于更多EMC信息，请参考以下应用笔记：

- AN1015 提高微控制器EMC性能的软件技术
- AN1709 EMC设计指南。

关于循环冗余校验（CRC）计算的更多详细信息，请参考以下应用笔记：

- AN4187：在STM32系列中采用CRC外设。

以下安全手册可从www.st.com获得：

- UM1741（适用于F0系列）
- UM1814（适用于F1系列）
- UM1845（适用于F2系列）
- UM1846（适用于F3系列）
- UM1840（适用于F4系列）
- UM1813（适用于L1系列）

2 软件包变化概述

表 1总结了STL软件包及所含的HAL固件。

表1. STL软件包概述

STM32系列	HAL驱动器	CMSIS驱动器	通用STL栈特定测试	所含的项目
F0	版本 1.5.0	版本 2.3.0	版本 2.1.0	SMT32052B-EVAL
F1	-	-	-	-
F2	版本 1.1.3	版本 2.1.2	版本 2.1.0	STM322xG_EVAL
F3	版本 1.3.0	版本 2.3.0	版本 2.1.0	SMT32373C-EVAL
F4	版本 1.6.0	版本 2.5.1	版本 2.1.0	STM324xG_EVAL
L0	版本 1.8.0	版本 1.7.0	版本 2.1.0	STM32L0xx_Nucleo
L1	版本 1.2.0	版本 2.2.0	版本 2.1.0	STM32L152D-EVAL
L4	-	-	-	-

固件采用通用的目录结构。它基于可用的驱动组，专用于给定产品或与特定的HW开发工具相关。部分驱动与整个STM32系列和ST调试有共同之处。

表 2详细说明了基础结构，以B类要求为目标的自检程序和方法收集在通用STL栈和产品特定STL栈目录下。根据用户应用HW，其余驱动大多都有特殊用途，以最终用户项目的变更或替代为准。

表2. FW结构的主要组织

目录	驱动	备注
驱动	BSP	评估板特定驱动
	CMSIS	内核特定驱动
	HAL	产品特定外设驱动
Utilities	CPU、字体、日志	通用调试/开发支持
中间件	通用STL栈	通用STM32 STL程序
项目/xxxxxx_EVAL	集成示例	依赖产品和工具的特定程序和评估板配置及集成示例
	产品特定STL栈	依赖产品和工具的STL程序和配置

特定STM32产品及专用评估板的所含项目已准备好，并在三大环境和工具链下进行了测试：

- IAR™-EWARM版本 7.80
- Keil®/MDK-ARM® 版本 5.06
- Ac6 Eclipse™ 版本1.8.0

表 3 和 表 4 分别总结了这些项目的详细结构及STL栈常见和特定部分内所含的文件列表。
表 5 列出了示例中所用的附加支持文件。

表3. 通用STL软件包的结构

STL	通用的STL栈源文件	
	文件	说明
启动测试	stm32fxx_STLstartup.c	启动STL流程控制
	stm32fxx_STLclockstart.c	时钟系统初始测试
运行时测试	stm32fxx_STLmain.c	运行时STL流程控制
	stm32fxx_STLclockrun.c	部分时钟测试
	stm32fxx_STLcrc32Run.c	部分Flash测试
	stm32fxx_STLtranspRam.c	部分RAM测试
头文件	stm32fxx_STLclassBvar.h	B类变量定义
	stm32fxx_STLlib.h	整体STL包括控制
	stm32fxx_STLstartup.h	初始过程STL头文件
	stm32fxx_STLmain.h	运行时过程STL头文件
	stm32fxx_STLclock.h	时钟测试头文件
	stm32fxx_STLcpu.h	CPU测试头文件
	stm32fxx_STLcrc32.h	Flash测试头文件
	stm32fxx_STLRam.h	RAM测试头文件

表4. 产品特定STL软件包结构

STL	产品特定STL栈源和头文件	
	文件	说明
源	stm32xxx_STLcpustartIAR.s stm32xxx_STLcpurunIAR.s stm32xxx_STLRamMcMxIAR.s stm32xxx_STLcpustartKEIL.s stm32xxx_STLcpurunKEIL.s stm32xxx_STLRamMcMxKEIL.s stm32xxx_STLcpustartGCC.s stm32xxx_STLcpurunGCC.s stm32xxx_STLRamMcMxGCC.s	汇编语言写的启动和运行时CPU和RAM测试代码， 分别针对IAR，Keil®和 AC6
数据头	stm32xxx_STLparam.h	产品特定STL配置文件

表5. 集成支持文件

支持集成示例STL实现的文件	
startup_stm32xxxxIAR.s	IAR™ 编译器的C启动
startup_stm32xxxxKEIL.s	ARM™ 编译器的C启动
startup_stm32xxxxGCC.s	Ac6编译器的C启动
main.c	示例的主流程
stm32xxxx_hal_msp.c	应用特定HAL驱动初始化
stm32xxxx_it.c	STL中断，时钟测量处理和配置程序
main.h	主流程头文件
stm32xxxx_hal_conf.h	HAL驱动配置文件
stm32xxxx_it.h	ISR头文件

3 STL软件包之间的主要差别（从产品的角度来看）

用户可发现一些小差别，这主要缘于产品的硬件偏差及编译器和调试工具的不兼容性。

本节对主要差别进行了说明，这些差别主要由不同STM32产品之间的兼容性造成，表 6对此进行了总结。



表6. 不同STM32微控制器之间的兼容性

特性	STM32F0	STM32F1	STM32F2	STM32F3	STM32F4	STM32L0	STM32L1	STM32L4
ARM® Cortex®-M4	M0	M3	M3	M4	M4	M0+	M3	M4
技术结点 (nm)	180	180	90	180	90	110	130 / 110	90
频率 (MHz)	48	24 至 72	120	72	168	32	32	32
性能 (DMIPS)	38	61	150	61	210	26	33	33
Flash存储器 (KB)	16 至 128	16 至 1024	128 至 1024	32 至 256	128 至 2048	32 至 192	35 至 512	35 至 512
Flash上的ECC	无	无	无	无	无	有	有	有
CRC (可配置)	有	无	无	有	无	有	无	有
RAM (KB)	4 至 16	4 至 96	64 至 128	16 至 48	64 至 256	8 至 20	4 至 80	4 至 80
RAM奇偶校验/加扰	有/有	否 (是)	无/无	有 ⁽¹⁾ / 有	无/无	无/无	无/无	有 ⁽¹⁾ / 无
辅助RAM	无	无	有	有 ⁽²⁾	有	无	无	无
数据EEPROM (KB)	-	-	-	-	2 至 16	2	2 至 16	-
EEPROM上的ECC	无	无	无	无	无	有	有	无
IWDG窗口选项	有	无	无	有	无	有	无	有
时钟系统 ⁽³⁾	HSI14、HSI48 (LSI~40kHz)	(LSI~40 kHz)	(LSI~32kHz)	(LSI~40 kHz)	(LSI~32kHz)	MSI、HSI48 (LSI~38kHz)	MSI (LSI~38)	MSI、HSI48 (LSI~32kHz)
时钟交叉参考测量	TIM14/Ch1	-	TIM5/Ch4	TIM14/Ch1 ⁽⁴⁾	TIM5/Ch4	TIM21/Ch1	TIM10/Ch1	TIM16/Ch1
时钟参考 下一个选项	GPIO、RTC、 HSE32、MCO	-	GPIO、RTC、 LSI、LSE	GPIO、RTC、 HSE32、MCO	GPIO、RTC、 LSI、LSE	GPIO、MSI、 LSI、HSE_RTC、 LSE	GPIO、RTC、 LSI、LSE	GPIO、RTC、 LSI、LSE、 MSI、HSE32、 MCO
扩展的V _{DD} 范围	有	有 ⁽²⁾	有	有 ⁽²⁾	有	有 ⁽⁵⁾	有 ⁽⁵⁾	有 ⁽⁵⁾

1. 所有产品上均未出现奇偶校验位。当有奇偶校验位时，它不能覆盖整个RAM，仅覆盖一部分。不含地址，除最新STM32F30x、STM32F333、STM32F301及STM32L4xx产品外。
2. 仅一些产品提供。
3. 所有系列成员均具备HSI16、HSE、PLL、LSI和LSE时钟源特征。表中列出了附加的时钟源。
4. TIM16/Ch1：用于STM32F30xx。
5. ULP，动态电压调节管理。

3.1 CPU测试

高级编译器无法访问一些特定的运行。这也是启动和运行时测试代码被写入汇编且已用编译器之间的助记符略微不同的原因。

由于STM32微控制器所用的Cortex® 内核之间的可用指令集各不相同，这些测试取决于产品。例如，由于ARM® Cortex®-M0+内核指令集受限，加载32位直接常量操作数的指令被加载那些存放在代码内存上的常量的指令所取代。

3.2 时钟测试和时基间隔测量

内部定时器用于交叉检查频率测量。当系统时钟由外部晶振或陶瓷谐振器提供时，需使用该方法测定谐波或分谐波频率，或在应用定时中检测任何明显的差异。不同的产品定时器专用于执行此类交叉检查测量。

当用于测量的中断向量跟所给定的芯片的具体定时器相依赖时，特定定时器的初始配置稍有不同。

一些老产品不支持交叉参考测量。

如果交叉检查测量取决于RC时钟，用户必须考虑整个温度范围内该时钟源的精确度。为防止任何时钟故障误检必须注意这一方面，尤其是当自检单元须在更广泛的温度范围内运行时。在监控环境温度趋势时，用户可采用适合的时钟测试算法，或考虑将更精确的时钟源作为时钟参考。

3.3 SRAM测试

确保数据字及其地址的单位冗余保护的硬件技术是遵循易失性内存相关标准的最低要求（不仅检测数据区域的错误，还检测其内部地址和数据路径的错误）。一些较老的ST产品不具备该硬件（部分或全部）冗余特征，因而应采用适当的软件方法间接满足这些要求，最好结合硬件考虑。

令人遗憾的是，执行这些测试使用了一部分微控制器运算能力，从而使得整个诊断测试变得更长。因此，软件方法仅可用于静态错误。即使非常复杂的测试也不能有效覆盖瞬时错误，因此，这些测试的诊断覆盖率基本上始终受限。

SRAM测试必须遵循拓扑模式。阵列中的逻辑邻位（属于单字）物理上彼此分离，而带后续逻辑地址的字对在物理上却共享邻位，此时可采用按字测试的方法。此外，当地址顺序不规则时（对于一些较老的STM32产品），在测试中必须遵循字的物理地址顺序（称为加扰）。

用户必须确保对应用产品执行与RAM设计相对应的适当测试。这通过汇编编译器的ARTISAN符号定义完成。对于STM32F0xx、STM32F1xx和STM32F3xx产品，必须对该符号进行定义。

用户可以通过定义USE_MARCHX_TEST来简化并加速运行时测试中用到的Marching C算法。如果该符号被定义，可跳过两个中间的推进步骤，无需在透明测试中执行（见第 6.4.7 节）。初始测试过程中，建议至少保持完整的March C算法。

一些ST微控制器具备内置单位冗余（硬件奇偶校验）的字保护功能，用于CCM RAM或至少用于部分SRAM。该硬件保护是IEC 60335和IEC 60730要求的可接受方法之一。另一方面，除最新产品外，如STM32F30x、STM32F333、STM32F301或STM32L4xx内置的奇偶校验功能不包括地址部分。

然而，当奇偶校验不包括地址时，明智的做法是引入一个软件March，这是因为其涉及内部地址和数据路径的附加测试。

应用软件March测试可用于测试那些未被硬件奇偶校验覆盖的部分RAM，或用作补充性测试方法。

通过应用间接测试技巧可增加SRAM内储存的信息可靠性，如物理分离区内的安全关键信息双重存储，其形式为两个倒置的模式（基于辐射会引起损坏或EMI通常会攻击有限的物理内存区段的事实）或者对这些数据区进行专门的校验签名。

硬件RAM奇偶校验是可选功能。启用该功能时，建议在代码执行开始时执行整个RAM的SW初始化，以避免在读取非初始化位置时出现奇偶校验错误（对于局部变量，当进行分配和读取时使用了不同的访问路径就是这种情况）。最好的方法是在启动过程中。在启动汇编码内插入一个简单循环可解决问题，并可初始化特定RAM区域内的奇偶校验系统：

```
; Program starts here after reset
;-----
Reset_Handler
; Parity system initialization has to be performed here prior to the
; startup self-test procedure
;-----
; r0 is used as a pointer to RAM,
; r1 keeps end address of the area
;-----
; At every step of the loop, the 32-byte block (r2-r9) is copied to RAM
; starting from address kept at r0, r0 is then increased by 32
; the loop body is performed while r0<r1
    LDR R0, =RAM_block_begin
    ADD R1, R0, #RAM_block_size
RAM_init_loop
    STMIA R0!, {R2-R9}
    CMP R0, R1
    BLT RAM_init_loop
```

```
; RAM is initialized now, program can continue by startup self-test
LDR R0, =STL_StartUp
BLX R0
```

注： 由于该循环的目的是初始化奇偶校验系统，因而由STMIA指令复制的寄存器实际内容与此不相关。RAM内容将在编译器标准启动程序进行初始化。RAM_block_begin和RAM_block_size常量设置应与STMIA指令复制的数据数量一致，以防止任何未定义的内存访问。

如果采用高级编译器，当在RAM栈区域执行初始软件March测试时，将会破坏所有栈内容，包括测试程序自身的返回地址。返回地址的存储和恢复程序取决于编译实现，并随优化级别不同而不同。除优化工作外，这是支持SRAM测试的程序以汇编的形式编写的主要原因，它独立于更高级的编译实现。另一方面，该解决方案会产生轻微的工具依赖性，因而必须使不同的汇编源文件正确通过其编译过程。

3.4 Flash存储器完整性测试

Flash存储器测试基于内置的HW CRC单元。一些STM32微控制器具备可配置单元，因而，初始化配置可稍有不同，然而，所用的多项式计算所有产品相同。

对于所有CRC不能配置的产品，用户应将 *stm32fxx_STLparam.h* 配置头文件中的宏定义 *CRC_UNIT_CONFIGURABLE* 注释掉。

存储CRC计算值的区域必须从计算范围排除。检查区域的边界必须与测试中用到的多重测试块的大小一致。根据stm32fxx_STLparam.h文件中定义参数FLASH_BLOCK_WORDS，块大小默认设为16字（64字节）。检查中未使用的内存区域可通过预定义值来识别。

非易失内存测试下的范围由用户界定。在运行时间中，如果整个内存范围的测试因太长而不能接受，用户可将其分成若干片段，这些片段与执行程序的本地区域相一致。这需要对被测试区域进行动态修改，以便在那些区域单独执行测试。

必须将CRC计算结果与相应的参考值进行比较，参考模式可由编译器（IAR™）自动提供或由在外部计算后由最终用户添加（MDK-ARM®和AC6）。

当工具本身不支持CRC模式时，在示例项目中提供特定的脚本文件（*crc_gen_keil.bat* or *crc_gen_gcc.bat*），以运行自动计算校验和的程序。它们取决于是否安装了Srecord GNU工具，<http://srecord.sourceforge.net>免费提供Srecord GNU工具。修改链接器输出提供的默认HEX文件，并将CRC参考值插入新的HEX文件。用户必须确保修改后的HEX文件（*output_name_CRC.hex*）用于应用下载（如执行*crc_load.ini*文件或当调试或下载会话开始时适当修改启动配置属性）。

测试Flash存储器完整性时，通过硬件CRC生成器完成的CRC计算大大降低CPU负载。测试Flash时可以使用DMA来传递数据。与软件测试相比，应用DMA时，CPU负载大大降低，但测试本身的速度不会改变这么多，这是因为DMA至少需要几个周期来读取和传输数据。当DMA服务于一些其他并行传输或中断时，测试甚至会减速。此外，测试初始化时还需要一些附加的DMA配置。用户可在专用应用笔记AN4187“使用STM32 系列中的CRC外设”中找到关于CRC计算DMA使用的详细信息。

3.5 启动和系统初始化

考虑到硬件偏差、专用调试工具及所用编译器，调试和诊断工具（如识别复位原因）的初始系统配置和设置之间存在差别。标准的产品启动文件也做了相应修改，在程序启动的最开始加入了一系列的启动检测程序。

3.6 固件配置参数

STL代码中用到的所有C代码中的STL配置参数和常量均收集入文件`stm32fxx_STLparam.h`中。配置差别涉及大小不同的测试区、不同的编译器和控制流的轻微偏差。

修改初始/运行时测试流程时，用户必须注意执行的控制流程可能出现的损坏。在这种情况下，互补控制流程计数器的数值可不同于针对流程检查点对比而定义的常量（见[第 5.1.3 节](#)）。为防止任何控制流程错误，用户必须通过适当的方式更改这些常量的定义。

对于专用的汇编编译器来说，有一些参数需要定义，更多详情可在[第 5.2 节](#)找到。

[表 7](#)总结了配置选项。



表7. 如何管理兼容性及配置STL软件包

特性	IAR™-EWARM	MDK-ARM®	GCC
ARM® Cortex® 内核	包括正确的启动和运行时CPU测试程序，正确处理内核故障和异常		
频率（MHz）	配置 <code>stm32fxx_STLparam.h</code> 中的SYSTCLK_AT_RUN_HSE/SYSTCLK_AT_RUN_HSI/HSE_VALUE/HSE_CLOCK_APPLIED/LSI_Freq参数		
Flash存储器（KB）	设置 <code>project icf</code> 文件中项目链接器选项菜单ROM_region 中的校验和选项。	设置 <code>stm32fxx_STLparam.h</code> 中的ROM_START、ROM_END 设置 <code>project sct</code> 文件中的LR_IROM1加载区 定义 <code>startup_stm32yyyyxxKEIL.s</code> 或 <code>project.sct</code> 文件中的Check_Sum	设置 <code>stm32fxx_STLparam.h</code> 中的ROM_START、ROM_END 定义 <code>project ld</code> 文件中的Flash存储器区域。
Flash上的ECC	通过中断或轮询方式执行ECC事件处理		
CRC（可配置）	处理 <code>stm32fxx_STLparam.h</code> 中的CRC_UNIT_CONFIGURABLE参数		
RAM（KB）	设置 <code>project icf</code> 文件中的RUN_TIME_RAM_BUF_region、RUN_TIME_RAM_PNT_region、CLASS_B_RAM_region、CLASS_B_RAM_REV_region、RAM_region。	设置 <code>project sct</code> 文件中的RAM_BUF、RAM_PNT、CLASSB、CLASB_INV RW_IRAM1 处理 <code>stm32fxx_STLparam.h</code> 中的RAM_START、RAM_END、CLASS_B_START、CLASS_B_END参数	定义 <code>project ld</code> 文件中的CLASSBRAM和RAM区。 处理 <code>stm32fxx_STLparam.h</code> 中的RAM_START、RAM_END、CLASS_B_START、CLASS_B_END参数。
RAM奇偶校验	通过中断或轮询方式处理RAM奇偶校验事件		
RAM扰码 ⁽¹⁾	当应用加扰时，定义项目编译器/处理器选项菜单中的ARTISAN=1	当应用加扰时，定义目标/Asm /条件运算选项汇编控制符号菜单中的ARTISAN=1	当应用加扰时，定义汇编属性/工具设置/MCU GCC编译器/综述/汇编器标记中的ARTISAN=1。
透明RAM测试过程中的March-X流程	当应用March-X流程时，定义项目编译器/处理器选项菜单中的USE_MARCHX_TEST=1	当应用March-X流程时，定义目标/Asm /条件运算汇编控制符号菜单中的USE_MARCHX_TEST=1	当应用March-X流程时，定义汇编属性/工具设置/MCU GCC编译器/综述/汇编器标记中的USE_MARCHX_TEST=1。
E2PROM上的ECC	通过中断或轮询方式执行ECC事件处理		
IWDG选项	处理 <code>stm32fxx_STLparam.h</code> 中的IWDG_FEATURES_BY_WINDOW_OPTION参数		
时钟交叉参考测量	设置适当的定时器系统，以进行交叉参考测量并处理其事件		
调试选项 ⁽²⁾	设置 <code>stm32fxx_STLparam.h</code> 中的STL_VERBOSE_POR、STL_VERBOSE、STL_EVAL_MODE、STL_EVAL_LCD参数		

1. 工具特定程序（写入汇编程序的源代码）。

2. 评估板特定程序 (不是安全码的组成部分, 但可用作应用集成示例)。



3.7 固件集成

项目示例提供了以B类要求为目标的自检程序和方法，显示如何将固件集成到一个真实的应用中。每个集成例子都采用专用产品和评估板。除通用驱动和程序外，还包括产品、评估板或编译器特定驱动，它们与安全任务没有直接关系，但可用于展示或调试（[第 2 节](#)说明了详情）。

用户必须负责专用链接器文件内容和项目特定设置，以将STL软件包和所有使用方法集成入目标应用。

定义被测试的内存区（RAM和Flash）、B类变量和栈内存空间分配及控制流程时应特别小心。

本文件的以下各节将提供更多详细信息。

3.8 HAL驱动接口

当从数据包中移除所有调试和冗余支持（UART信道、LCD显示器、LED）时，HAL层和STL程序之间的接口下降至需控制启动和运行时自检期间使用的特定外设。[表 8](#)给出了概述。

表8. 所有STL栈程序使用的HAL驱动概述

HW组件	STL文件中	使用的HAL驱动
内核SysTick定时器	HAL_SYSTICK_Config	stm32xx_STLmain.c
NVIC	HAL_NVIC_SetPriority HAL_NVIC_EnableIRQ HAL_NVIC_SystemReset	stm32xx_STLstartup.c stm32xx_it.c
时钟系统	HAL_RCC_OscConfig HAL_RCC_ClockConfig HAL_RCC_EnableCSS	stm32xx_STLstartup.c stm32xx_STLclockstart.c stm32xx_STLclockrun.c
定时器	HAL_TIM_IC_Init HAL_TIMEx_RemapConfig HAL_TIM_IC_ConfigChannel HAL_TIM_IC_Start_IT __TIMx_CLK_ENABLE	stm32xx_it.c
CRC单元	HAL_CRC_Init HAL_CRC_DeInit HAL_CRC_Accumulate HAL_CRC_Calculate __HAL_CRC_DR_RESET __CRC_CLK_ENABLE()	stm32xx_STLstartup.c stm32xx_STLcrc32Run.c

表8. 所有STL栈程序使用的HAL驱动概述（续）

HW组件	STL文件中	使用的HAL驱动
IWDG & WWDG	HAL_IWDG_Init HAL_WWDG_Init HAL_IWDG_Start HAL_WWDG_Start HAL_IWDG_Refresh HAL_WWDG_Refresh __HAL_RCC_CLEAR_FLAG __HAL_RCC_GET_FLAG __WWDG_CLK_ENABLE()	stm32xx_STLstartup.c stm32xx_STLmain.c
HAL层	HAL_Init HAL_IncTick HAL_GetTick	stm32xx_STLstartup.c stm32xx_STLmain.c stm32xx_it.c

4 符合IEC, UL和CSA标准

IEC（国际电工技术委员会）是全球认可的非营利性非政府机构，致力于为广泛的电气、电子和相关技术编制和出版国际标准。IEC标准主要针对安全和性能、环境、电能效率及其可再生能力。IEC与ISO（国际标准组织）和ITU（国际电信联盟）密切协作。其标准为硬件及软件解决方案提供建议，根据应用目的，这些标准可分为若干安全等级。

在电子标准领域内，其他全球公认的机构有德国的TUV或VDE、英国的IET、美国和加拿大的IEEE、UL或CSA。除在标准开发期间提供专业知识外，这些机构还可担任测试、检验、咨询、审计、教育和认证机构。它们之中的大部分机构都以全球市场准入为目标，但其首先是作为本地国家认证机构（NCB）或国家认可测试实验室（NRTL）被认可和注册。这些机构的主要目的是为电器制造商提供标准依从性和质量测试服务。

由于全球化进程，很多制造商努力争取国家标准的协调。这与很多政府的努力恰恰相反，它们仍通过建立行政壁垒来保护小规模的本地制造商，以防止国外厂商轻松进入本地市场。事实上，很多标准都十分协调，只是存在一些微小的差异。这使得认证流程更容易，与本地公认的任何一家机构合作都富有成效。

关键的IEC标准为IEC 60730-1和IEC 60335-1，与第四版开始的UL/CSA 60730-1和UL/CSA 60335-1相（此外，先前的UL/CSA采用了UL1998标准的参考）协调。这些标准涉及家庭和相似环境内家用电器的安全性。

内置有电子回路的家用电器应经过组件故障测试。这里的基本原则是，如果出现任何组件故障，家用电器必须保持安全。从这一点来看，微控制器是和任何其他组件一样的电子组件。如果安全取决于电子组件，则其必须在连续两次出现故障后保持安全。这意味着，出现一个硬件故障且微控制器不运行（复位或不能正常运行）时，家用电器必须保持安全。

此外，如果安全还取决于软件，这被认为是第二次应用故障，需要符合ClassB或者ClassC软件要求。所需条件在IEC 60335-1标准的附件Q和R及IEC 60730-1标准的附件H中详细说明。

三个分类定义如下：

- **A类：**安全不依赖SW
- **B类：**SW防止不安全操作
- **C类：**SW旨在防止特殊危害。

该应用笔记范围及相关STL软件包为B类规范。

符合C类标准的家用电器不在本文件范围内，这是因为它们需要经过更稳定的测试，且通常会需要一些特定的HW冗余解决方案，如双重微控制器操作。在这种情况下，用户应使用产品专用安全手册，并应用手册中说明的方法。

微控制器的B类标准与硬件和软件相关。符合标准的部件可分为两组，即MCU相关和应用相关，如 表 9中例示。

应用相关部分依赖客户应用结构且必须由用户定义及开发（通信、IO控制、中断、模拟输入输出），微型部件仅与微结构相关，可为通用类（内核自诊断、易失和非易失内存完整性检查、时钟系统测试）。这组MCU相关的测试专注于ST方案，它基于强大的STM32微控制器硬件特征，如双重独立看门狗、CRC单元或系统时钟监控。

表9. 必须在B类标准依从性下测试的MCU部件

产品部	根据标准进行测试的组件
MCU相关	CPU寄存器
	CPU程序计数器
	系统时钟
	常量和变量内存
	内部寻址（和外部寻址，如有）
	内部数据路径
应用相关	中断处理
	外部通信
	时序
	I/O外设
	模拟A/D和D/A
	模拟复用器

4.1 STL固件包所含的通用测试

经过认证的STM32 STL固件包由下列软件模块组成：

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

注：上述列表的最后两项标准中未明确要求，但其提高了整体故障检测覆盖率，涵盖部分特定的必要测试（如内部寻址、数据路径、定时等）。

表 10说明了MCU测试（以下章节将详细说明）所用的方法概述。

用户可将部分或所有已认证SW模块纳入其项目。如果它们不发生变化, 并根据这些指南而进行集成, 获得认证终端应用所需的时间和成本将大幅下降。

当测试取消时, 用户应考虑副作用, 这是因为任何未应用的组件测试都可在其他组件的间接测试中起到重要作用。

表10. 相关ST软件包MCU相关测试中采用的方法

待验证组件	使用的方法	IEC 60730参考
CPU寄存器	所有寄存器和标记的功能测试均在启动时完成, 包括R13(栈指针)、R14(链接寄存器)和PSP(进程栈指针)。运行时的测试不检测R13, R14, PSP和标志。测试栈指针是否上溢, (下溢通过非直接方法检查) 链接寄存器由PC监控进行测试。如果发现任何错误, 软件直接跳至故障保护程序。	H.1.1.1 H.2.16.5 H.2.16.6 H.2.19.6
程序计数器	当程序计数器丢失或终止时, 两个不同的看门狗可复位设备, 它们在两个独立的时钟源运行。窗口看门狗由主振荡器驱动, 可执行时隙监控, 独立看门狗由内部低速RC振荡器驱动, 一旦启用将无法禁用。程序控制流程可采用专用软件方法监控。	H.1.1.3 H.2.18.10.2 H.2.18.10.4
寻址和数据路径	该组件由RAM功能和Flash完整性测试、栈上溢间接进行测试(将特定模式写在栈空间的下边界, 并定期检查是否存在损坏。为检查是否出现下溢, 可将第二个模式写在上边界, 如果其不在RAM顶部)。此外, 如果内存访问出现故障, CPU可获取BUS错误异常向量。	H.1.4.3 H.1.5.1 H.1.5.2 (仅限间接测试)
时钟	采用两个独立时钟源交叉检查来进行测量。被测量的频率控制定时器时钟, 另一个来给定定时器时钟输入。例如, 使用内部低速RC振荡器时基可检测外部晶振(谐波/次谐波)的错误频率。	H.1.3 H.2.18.10.1 H.2.18.10.4
常量内存	完整内存的32位CRC校验和测试在启动时完成, 部分内存测试在运行时间重复进行(逐块)。使用快速内置32位CRC计算单元。	H.1.4.1 H.2.19.4.1 H.2.19.8.1
变量内存	March C- 完整内存测试在启动时完成, 部分内存测试在运行时重复(在B类存储区域逐块进行)。在耦合故障优化覆盖的测试中, 遵循RAM内物理地址的扰乱次序。Faster March X可用于在运行时间进行测试。可采用双重冗余字保护(在非邻近内存空间保存相反值)的方法对安全相关的B类变量进行保护, 运行期间不对A类变量空间、栈及未使用空间进行测试。	H.1.4.2 H.2.19.6.2 H.2.19.8.2

应用测试主要用于检测永久性故障(覆盖d.c.故障模型下的故障)。由于相对长的重复测试周期, 任何软件测试进行的瞬时故障检测总是受到限制(与具备任何永久性检查能力的HW方法相比), 并被间接测试部分覆盖。

注: 如果模块发生微小变化, 用户应对所有模块进行记录, 在源文件内放入清晰的解释说明, 并将与认证程序的差别通知给认证机构。

4.2 应用相关特定测试不在ST固件自检库范围内

用户应当专注不包括在ST固件库内的所有应用相关的测试:

- 模拟部件测试 (ADC / DAC, 复用器)
- 数字I/O测试
- 外部寻址
- 外部通信
- 定时和中断。

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。建议从应用设计早期就开始遵循建议的测试原理。

该方法经常会带来HW和SW层级的冗余。

HW方法应基于:

- 输入和/或输出的倍增
- 参考点测量
- 模拟或数字输出时的回送读取控制, 如DAC、PWM、GPIO
- 配置保护。

SW方法应基于:

- 及时重复、多重采集 (在不同时刻和 (或者) 通过不同的方法, 测量, 判断或计算两次)
- 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)。

4.2.1 模拟信号

测量值应该以正确的时间间隔进行检查, 如果在应用中使用了模拟复用器, 则可结合模拟复用器测试, 使用空闲通道读取一些参考电压。此外, 还应检查内部参考电压。

一些STM32设备具备两个 (有些甚至有三个) 独立的ADC块。为了安全, 可用两个不同的ADC块执行同一通道上的转换, 这一点很有意义。对一个通道的多重数据采集或对比冗余通道, 然后进行平均操作。

4.2.2 数字I/O

B类测试必须检测数字I/O上的任何故障。可以通过同时对其他部件的进行检测的似真检测来实现（例如，当加热（或者制冷）数字控制打开（或者关闭）时，应该检查温度传感器的模拟信号变化）。可通过将正确的锁定序列应用于GPIOx_LCKR寄存器内的锁定位来锁定选择的端口位，从而防止端口配置出现意想不到的变化。在这种情况下，仅可在下一个复位序列进行重新配置。此外，位带特性可用于SRAM和外围寄存器的原子操纵。

4.2.3 中断

应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，采用一组增量计数器。每个中断事件使特定定时器递增。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。

配置锁功能可用于通过由TIMx_BDTR寄存器控制的三个层级来保护定时器寄存器设置。未使用的中断向量应转移至通用错误处理器。如果能简化应用中中断方案，对于非安全相关问题最好进行轮询。

4.2.4 通信

通信数据交换的正确性可以通过加入冗余来进行检查。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。必须对通信事件的周期性和发生率及协议错误信号进行永久性检查。

用户可在产品专用安全手册中找到更多信息和方法。

4.3 安全生命周期

对于UL/IEC 60730-1要求（涉及第H.11.12.3节所关注的系统错误预防），提供固件开发和维护。所有相关进程均遵循ST内部策略，从而确保其拥有必要的质量水平。

这些内部规则的应用及对公认标准的依从性是公认的外部检验机构进行定期检验和审计的目标。

涉及以下阶段：

安全要求规范

主要目标通过内部规划指明，以为用户应用提供独立的通用模块组，从而使其更容易集成入用户固件，其目的是符合UL/IEC 60730-1和UL/IEC 60335-1标准。使用已被认证机构认证过的方案和方法加速了用户开发和认证过程。

架构规划

STL软件包结构来源于长期的FW重复认证经验，模块在这里集成入ST标准外围库（在过去专用于不同的产品）。根据针对整个STM32系列而开发的新颖独特的硬件抽象接口（HAL），新FW的主要目标是去除任何对不同产品的硬件依赖和将安全相关部分集成到一个单个的通用的自检栈中。

从系统的观点来讲，在现有产品之间迁移或迁入新产品时，这种通用构造被认为更安全，维护和方案集成更简单。在很多不同的配置中，很多用户都使用相同的结构，因而其反馈绝对重要，有助于有效处理薄弱环节（如有）。

模块规划

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

编码

编码基于内部ST策略定义的原理，遵循公认的国际编码标准、验证工具和编译器。

强调执行非常简单、单一和透明的线性编码结构，在使用简化而清晰的输入和输出时，逐步调用定义的测试功能组。

流程由特定的流程控制机构提供保护，并与系统节拍中断服务同步，从而提供特定的特殊后台检测。一旦所有的局部检查程序成功完成，应专门提供硬件看门狗服务。

测试模块

已通过不同的开发工具对不同产品的功能性进行了模块测试。可在以下章节和专用于认证机构的特定测试文件中找到详细信息。

模块集成测试

已通过不同的开发工具对几个示例的不同产品进行了模块集成测试，主要针对适当的定时测量、代码控制流程、栈使用和其他方法。此外，也可在以下章节和测试文件中找到详细信息。

维护

对于FM维护，ST根据标准的内部流程采用客户反馈。定期或识别一些重大错误时公布新的升级服务。所有版本都通过描述解决方案及其集成的适当文件公布。升级之间的差别、应用修改及已知限制都在相关发布说明中描述。

特定工具用于支持适当的SW版本编号和源文件归档。

所有固件及文件均可通过www.st.com直接获取，也可向本地支持中心索取。

5 B类软件包

本节强调了ST解决方案中通用的基本原理。并对工程结构以及配置和调试进行说明。各个开发环境之间的差别（IAR™ EWARM, Keil® MDK 和AC6Eclipse™）也进行了说明。

5.1 使用的通用软件原理

ST解决方案内所用的基本软件方法和通用原理在本节详细说明。

5.1.1 故障安全模式

当自检程序检测到故障时，专用程序*FailSafePOR()*被调用。例程在*stm32xx_STLstartup.c*文件开始被预先定义。该程序的目的是提供唯一的输出并允许用户立即做出反应。

默认情况下，程序内部没有特定的处理，除调试支持和等待看门狗复位的空白循环外（调试模式下可防止复位）。该程序的内容应该由用户自行开发，在这里执行所有必须的操作，以保证应用处于一个安全的状态。同时根据所发现问题的严重程度对下一周期作出决定。

从程序的不同位置进行调用，以识别问题的严重程度并简化程序内部的决策流，用户可视情况重新定义程序，并传送一个特定的输入参数（简单的常量）。

[第 5.4节](#)中说明的调试或详细模式用于识别发生的错误。

5.1.2 安全相关变量和栈边界控制

强烈推荐以特定方式处理系统安全相关的临界值。

每个B类变量都作为一对互补值而储存在两个单独的RAM区。正常值和冗余互补值始终放在不相邻的内存位置。在这些RAM区，特定的中断子例程逐步连续执行部分RAM March C-或March X测试。用于测试区域临时储存和反向恢复的缓冲区在永久测试范围内。

每次使用数值前，用户必须确保比较每一个数值对的完整性。如果发现任何数值对的完整性被损坏，应调用故障保护模式。如果变量值变化，需要更新存储位置以保持正确的数值对完整性。

[图 1](#)是一个RAM配置示例。用户可根据应用需求和硬件能力修改RAM空间分配。为使运行时测试获得较好的一致性，所有B类区域均可并入一个单一的紧凑内存位置。

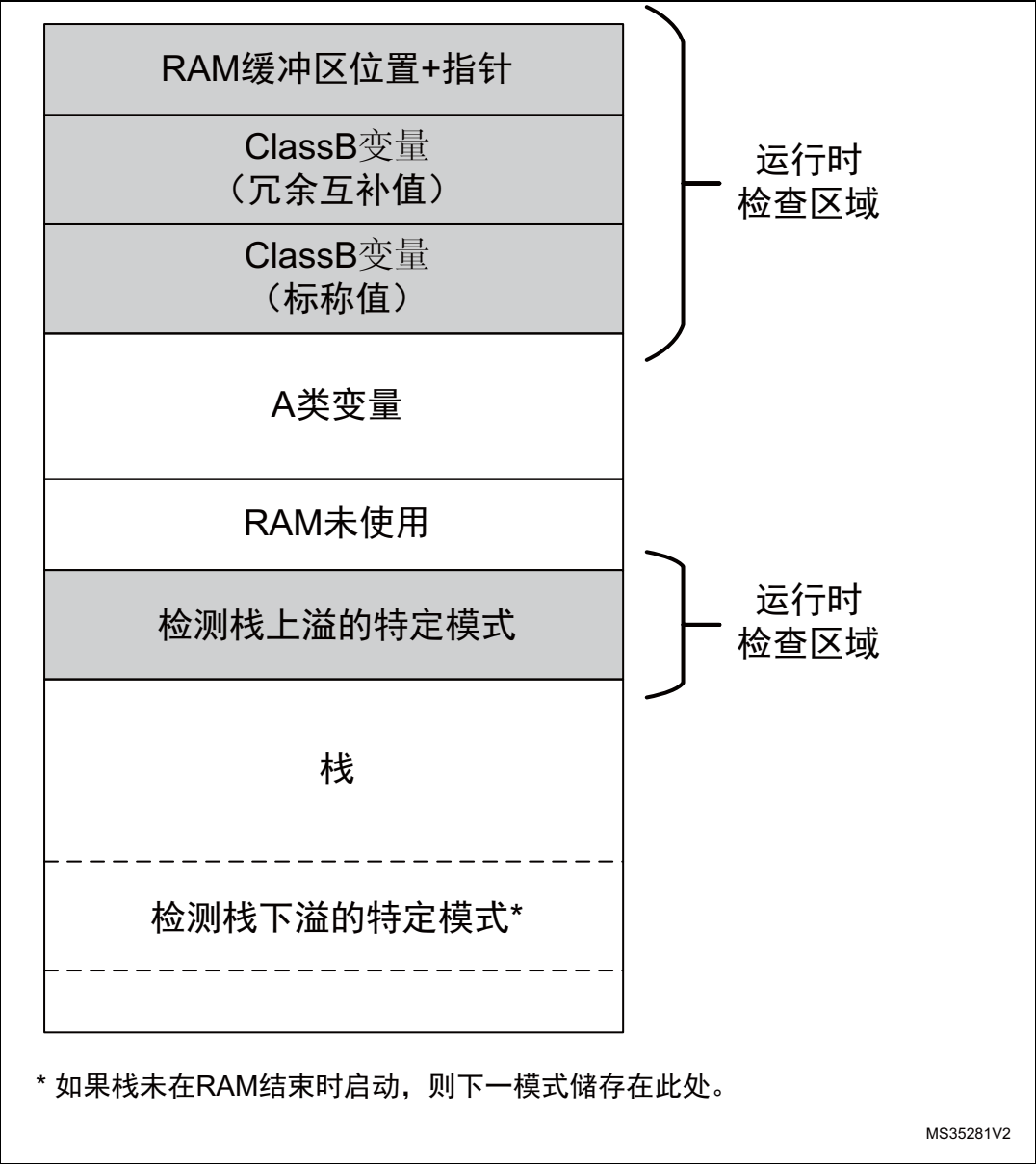
当第一步和最后一步测试重叠时，用户必须调整测试区域的大小，以增加单个步骤，包括地址加扰。在测试区域开始和结束，为块重叠和单步对齐留出相应空间是一个较好的主意。若需更多信息，请参见第 6.4.7节。

特定数据模式写在栈空间的下边界，并定期检查是否有损坏，以检测栈上溢。为检查下溢，如果栈不放在RAM顶部，可在上边界写第二个。如果栈放在RAM顶部，则下溢事件会引起硬件异常。

当应用使用多个单栈区域时，建议使用边界数据模式分开相邻区域，以检查所有指针是否仅在其专有区域运行。

运行时，栈和非安全RAM区不进行检查。

图1. RAM存储器配置示例



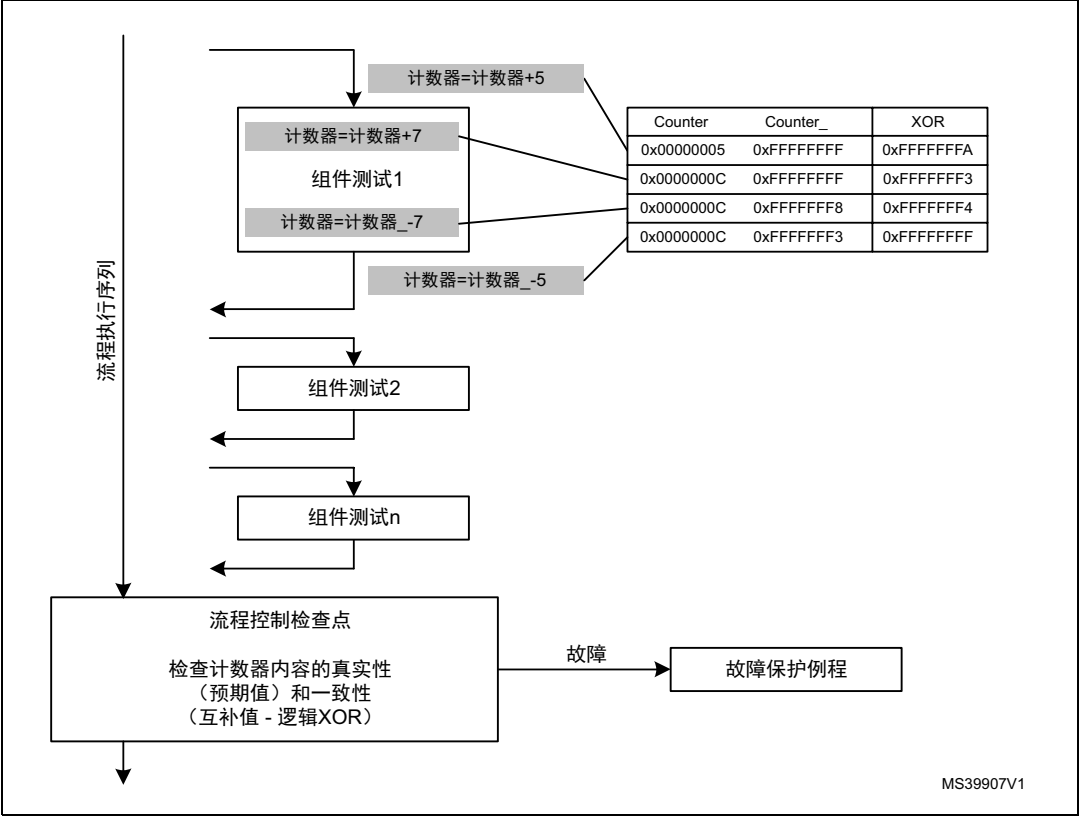
5.1.3 流程控制程序

程序流程控制是标准强烈推荐的方法，这是因为它能有效确保所有特定代码能正确执行并通过。

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

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

图2. 控制流程四步检查原理



注： 在该例子中，主层级组件测试1的调用点唯一编号被定义为5，程序本身的唯一编号被定义为7。为简单起见，计数器初始值为0和0xFFFFFFFF。图 2 右上角表格显示的是执行最后一步检查策略后计数器在四个步骤中如何变化及一致的互补状态（从程序返回）。

5.2 库的工具特定集成

本节说明如何根据不同的使用工具组织ST解决方案。

5.2.1 软件包所含项目

FW包括支持STM32产品各种专用评估板的实现示例。针对每个支持的开发板，都提供了以下三个IDE的工程：IAR™-EWARM、Keil® MDK-ARM® 和Ac6 Eclipse™。

建议检查并根据不同的工具进行正确的操作：

- 对于特定的STM32系列成员及使用的评估板，必须配置相应的Project.eww、Project.uvproj 或 .cproject项目文件。必须在预处理器设置部分进行正确的声明。
- 必须检查链接器脚本文件的*.icf（IAR™）、*.sct（Keil®）和*.id（Ac6）模板以确保包括B类RAM区域在内的所有的内存区域已经定义好。对于安全临界变量，第 5.1.2 节说明了RAM区的一致性程序，对于CRC Flash完整性，见第 3.4节。
- 进入主函数前，必须修改编码内标准的startup_stm32xxx.s文件（IAR™ 和Ac6编译器）或\$Sub\$\$main() 程序，以在程序流程开始时插入STL_StartUp()程序（对于Keil®编译器，见stm32fxx_STLstartup.c）。如果所有启动测试均通过，则调用宏GotoCompilerStartup()（在stm32fxx_STLparam.h文件中定义），以继续标准C启动程序。对于Keil® 调用 __main()，IAR调用程序__iar_program_start()；对于Ac6，调用程序Startup_Copy_Handler()。对于Keil® 编译器，采用了一点小小技巧，以决定是否必须调用启动测试程序或主流程。发出启动测试设置完成信号的特定模式储存在CRC单元独立的数据寄存器内（见stm32fxx_STLstartup.c文件）。
- 对于IAR，CRC的产生可以通过IAR的工程配置项进行使能和设置计算范围，对于Keil®和Ac6,需要根据第 3.4节中描述的具体方法进行计算。应在stm32xxx_STLparam.h文件中定义正确的常量

表 7： 如何管理兼容性及配置STL软件包第 16页给出了总结。

5.2.2 启动文件

为了运行自检程序的初始设置，为每一个项目准备特定的启动文件，以此作为设备复位后的第一个任务。自检启动例程不改变也不禁用编译器标准C启动文件。在完成启动测试后不久，以常规方式对变量和栈/堆阵进行初始化。

5.2.3 定义新的安全变量和受检内存区

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

A类和B类变量区域大小可在链接器配置文件中修改。新的B类变量必须在**stm32xx_STLclassBvar.h**头文件中声明，可采用以下IAR™语法：

```
__no_init EXTERN uint32_t MyClassBvar @ "CLASS_B_RAM";
__no_init EXTERN uint32_t MyClassBvarInv @ "CLASS_B_RAM_REV";
```

可采用以下 Keil®语法：

```
EXTERN uint32_t MyClassBvar __attribute__((section("CLASS_B_RAM"), zero_init));
EXTERN uint32_t MyClassBvar Inv __attribute__((section("CLASS_B_RAM_REV"), zero_init));
```

可采用以下Ac6语法：

```
extern uint32_t MyClassBvar __attribute__((section(".class_b_ram")));
extern uint32_t MyClassBvarInv __attribute__((section(".class_b_ram_rev")));
```

stm32fxx_STLclassBvar.h头文件、链接器配置文件及自检库配置文件

stm32fxx_STLparam.h中的变量定义之间必须始终保持一致，以对齐安全临界变量位置，内存范围区域的定义在启动及运行时SRAM测试期间测试。当使用Keil®和Ac6环境时，RAM/ROM区域开始和结束地址不导出。必须由用户在stm32xx_STLparam.h文件中对这些地址进行修改，stm32xx_STLparam.h文件包含执行正确的透明RAM和ROM（Flash）测试时所需的特定地址和常量定义。

SRAM测试程序被写入汇编并收集在编译器特定stm32xxxx_STLRamMcMxyyy.s汇编文件中。将程序集成入项目时，必须考虑产品的不同和汇编配置参数（ARTISAN和USE_MARCHX_TEST）。通过参数调用程序，确定测试区域的开始和结束并检查模式。用户必须按照测试应用范围遵循可用的物理地址空间。

当SRAM设计具备物理地址加扰特征时，必须定义ARTISAN参数，用户内存分配必须遵循加扰的地址模式重复。当测试块的间隔大小受到模式大小限制时，marching测试的内存块的起点必须与物理模式的起点对齐。

对于编译器层级的非易失内存检查来说，CRC模式计算的实现及被测试区域的定义各不相同。IAR™编译器的设置将与STM32硬件兼容的32位CRC结果直接纳入代码（见IAR™文件），Keil®和Ac6编译器不完全支持CRC计算。这就是STM32内部CRC生成器提供的结果不能直接用于这些项目的内存检查的原因。

用户必须采用一些其他的后置方法将正确的校验值写入代码（见[第 3.4 节：Flash 存储器完整性测试](#)中的更多详情），或忽略CRC计算进程输出的否定的比较结果。

注：最新版本的IAR™ 编译器包括配置校验和计算结果时的不兼容问题。这就是专用ST HAL驱动不能用于将数据储存在CRC单元的原因。当另外实现的__REV固有功能颠倒输入数据字节的顺序时，必须采用直接访问CRC_DR的方式来处理该问题。在以后更新编译器时，用户可期待对其进行修正。然后，当正确馈送CRC输入数据时，用户可切换回标准的HAL驱动调用（stm32fxx_STLstartup.c和stm32fxx_STLmain.c文件中的HAL_CRC_Calculate()或HAL_CRC_Accumulate()）。

5.2.4 应用实现示例

项目中main.c file文件提供一个短小的用户应用演示示例（见[第 6 节](#)）。它演示了如何将ClassB程序集成到一个具体的应用方案中去。

当使用条件编译时，用户可为了展示或调试而纳入一些控制专用评估板硬件的附加软件。这部分代码与安全无关，应从最终应用代码中删除。它使用以下硬件：

- 用于演示的LCD显示器（如评估板上有）
- 专用UART Tx端口（向超级终端窗口发送文本信息）
- 驱动LED的专用输出（表明软件例程被正确执行）。

5.3 执行定时测量和控制

专用I/O可用于在启动和运行时执行程序的定时测量。

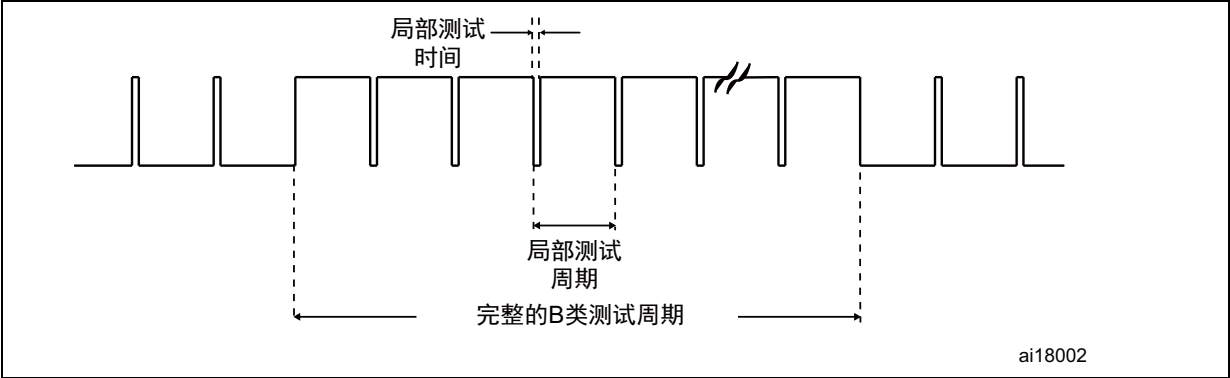
在单次运行中执行启动测试，测试时间取决于MCU性能和测试区的大小。运行期间的测试逐块执行，因此其时间也取决于被测试块大小和测试的重复频率。

用户必须在运行应用所需的性能和测试运行硬件的性能之间找到平衡。主要挑战在于实现较短的总体诊断测试间隔，同时保持应用过程安全时间在可接受的长度范围内。在关键情况下，运行时测试可限制在收集关键代码或数据的区域。局部测试间隔来自SysTick 1 ms间隔，可根据参数SYSTICK_10ms_TB默认设为10毫秒。

当将其缩短时，用户应考虑到该间隔也用于在运行期间计算系统时钟和LSI之间的时钟交叉参考率，因而其长度不得低于时钟交叉测量（默认设为8）所用的LSI周期数相应间隔。

当监控这些与板上LED相连的I/O信号时，应采用特定原理，如[图 3](#)所示：

图3. 诊断LED定时信号原理



当特定测试（RAM、Flash、时钟）激活及每次完成测试程序（启动时和运行期间）时，专用LED可状态切换。其中，在启动时和运行期间，这些LED信号可用于测量测试长度、局部测试的频率及完成整个单测试周期所需的时间。

完全检测被测试的专用区域后，一组局部测试又重新开始。

示波屏截图显示启动时及运行期间监控专用I/O信号的典型波形，分别如图 4和图 5中所示。

图4. 启动期间典型的测试时序

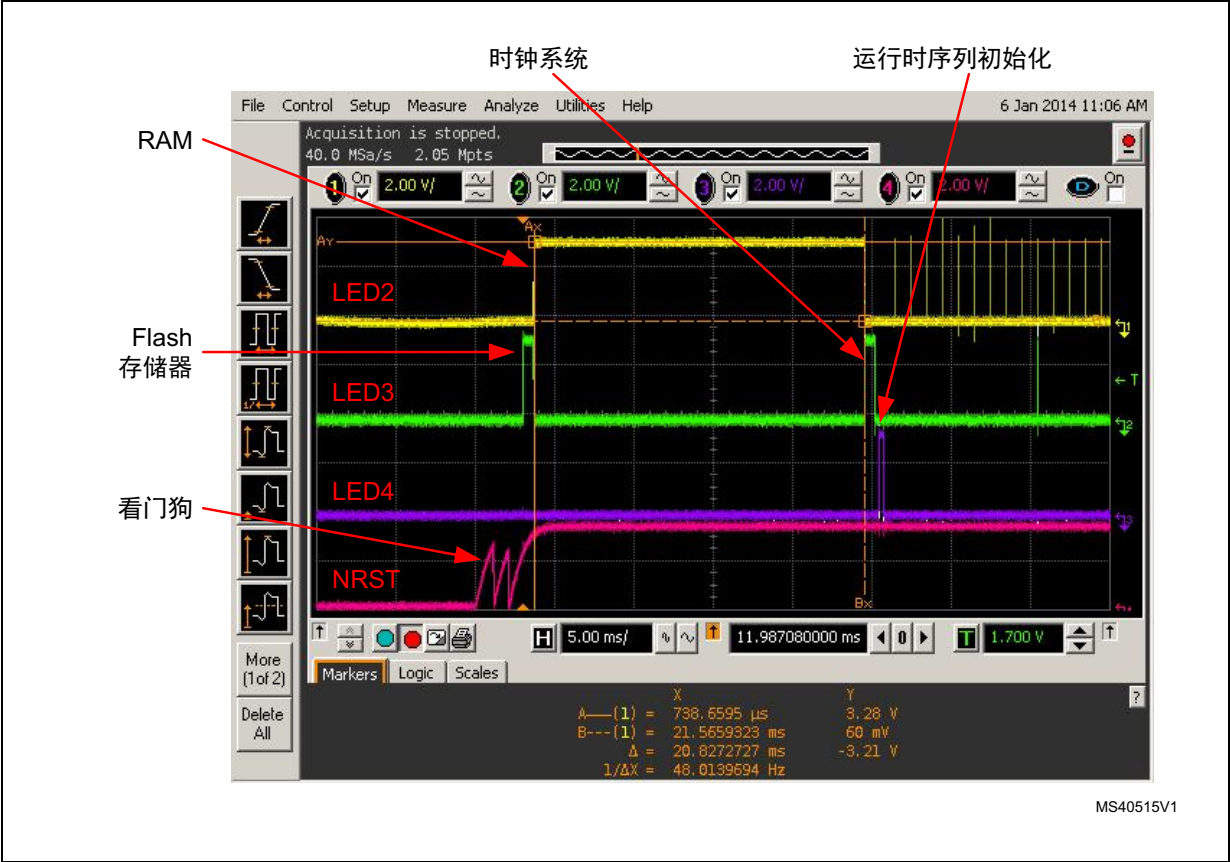


图5. 运行期间典型的测试时序

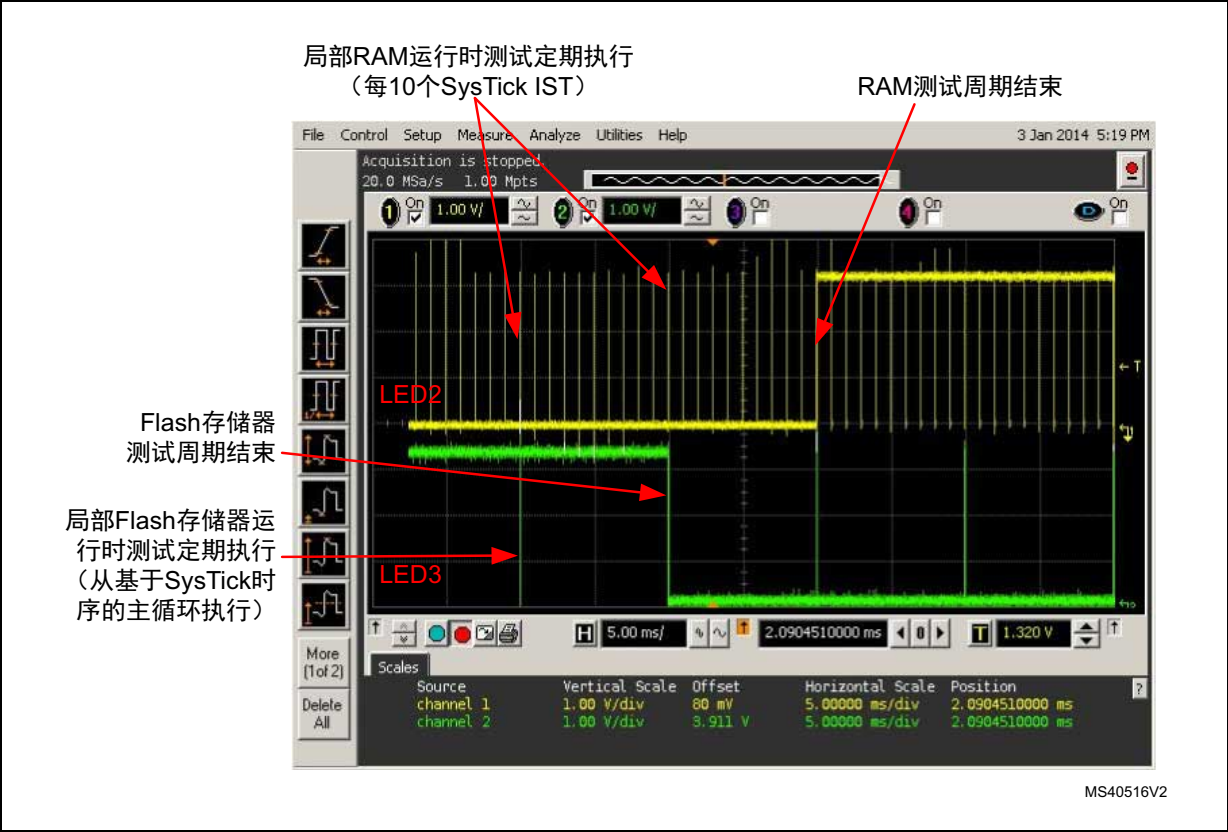


表 11总结了I/O信号，同时表 12列出FW（固件）包测试期间测定的典型值。

表11. 用于定时测量的信号

信号	STM32052B-EVAL	STM32 20-21-45-46G-EVAL	STM32373C-EVAL	STM3241G-EVAL	STM32NUCLEO L053R8	STM32L152B-EVAL
橙色LED2 (黄色)	PD9 (CN8-6)	PG8 (CN3-22)	PC1 (CN14)	PG8 (CN3-22)	PA5 (CN5-D13)	PD7 (CN10-15)
红色LED3 (绿色)	PD10 (CN8-4)	PI9 (CN1-12)	PC2 (CN14)	PI9 (CN1-12)	PA6 (CN5-D12)	PG14 (CN10-6)
绿色LED4 (紫色)	PD11 (CN8-2)	PC7 (CN3-20)	PC3 (CN14)	PC7 (CN3-20)	PA7 (CN5-D11)	PG15 (CN10-5)

表12. 结果对比⁽¹⁾

特性	STM32F0	STM32F2	STM32F3	STM32F4	STM32L0	STM32L1
XTAL (MHz)	8	25	8	25	8	8
PLL频率 (MHz)	48	120	64和72	168	32	32
Flash存储器测试, 启动 (毫秒), IAR™ / Keil® / Ac6	7.2 / 1.8 / 0.6	1.5 / 0.4 / 0.2	6.4 / 1.6 / 0.7	1.2 / 0.3 / 0.15	8.4 / 2.0 / 0.86	6.8 / 2.4 / 1.0
测试Flash存储器 (KB), IAR™ / Keil® / Ac6	41 / 10 / 10	32 / 9 / 9	37 / 12 / 11	32 / 9 / 9	32 / 12 / 10	35 / 12 / 10
完整RAM测试, 启动 (毫秒)	4.5	12.0	8.1	8.6	3.8	12.2
测试的RAM (KB)	8	128	32	128	8	32
时钟测试, 启动 (毫秒)	1.0	0.9	0.8	0.7	1.6	1.4
单个RAM测试, 运行时间 (μs)	19	6.3 / 6.0 ⁽²⁾	17	5.5 / 4.6 ⁽²⁾	30	25
单个Flash存储器测试, 运行时间 (μs)	30 / 24 ⁽²⁾	8.2 / 7.2 ⁽²⁾	19 / 17 ⁽²⁾	9.5 / 5 ⁽²⁾	40	29

1. 该表中的数值为参考值, 进行测量时无需调试支持 (除LED控制外)。这些数据可随编译器的优化设置变化而变化, 其他软件包配置取决于待检查的区域。

2. Ac6.

5.4 软件包配置和调试

STL软件包的配置必须遵循实际产品的正确设置。

有时候, 一个简单的应用结构就涉及暂停或去除STL软件包的功能部分 (系统使用内部时钟)。相反, 一些功能可在软件包调试期间临时添加, 这是因为它们可以为这一阶段的开发者提供帮助。

本节说明如何配置、修改和调试ST解决方案。

5.4.1 配置控制

在两个基本层级完成软件的配置:

- 项目设置: 设置必须遵循不同STM32产品之间的差别。这一部分主要由适当的项目配置及相关配置文件自动完成。
- 用户设置: 主要集中在B类配置文件 *stm32xx_STLparam.h* 中。该文件中定义的一组常量可控制一些函数的条件编译。对这些常量进行配置以正确运行所有测试。

根据终端应用, 一些运行时测试可跳过。如果测试的周期性和使用频率一致, 那么开机测试即可, 可避免进行运行时测试 (例如, 洗衣机就是这种情况: 用户在每次使用时都开启/关闭应用)。这一点必须由选定的测试机构进行具体问题具体分析。

为获得最大稳定性，当应用开发在结束阶段时，建议使用硬件选项字节启用独立的看门狗，并在主例程中尽可能启动窗口看门狗。默认情况下，STM32自检库演示中并不执行。

建议在测试结束阶段执行窗口功能，在调试时冻结看门狗选项。

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

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

如果32位CRC检查由STM32内部CRC生成器执行，则有助于减少运行期间的CPU负载（32位宽CRC计算使用标准的0x04C11DB7多项式）。与所有后续运行时间检查进行对比时，经过验证的32位CRC值可另存为参考值。

Keil® 和Ac6环境不支持CRC生成；调用CRC检查程序与不正确的CRC校验值进行对比将导致应用不断复位（现象与检测到故障类似）。

如果用户在启动时想临时禁用CRC检查，最容易的方法是修改比较CRC计算结果的逻辑运算（假设计算的CRC不同于参考值）。

在`stm32xx_STLstartup.c`文件中修改：

```
if(crc_result != *(uint32_t *)&REF_CRC32))
```

为：

```
if(crc_result == *(uint32_t *)&REF_CRC32))
```

5.4.2 冗长诊断模式

在冗长模式下，专用USART Tx串行外设线用作B类状态文本信息的标准输出。当该串口线可以由外部终端监控时，该模式在调试阶段有用（设置为115200 Bd，无奇偶校验，8位数据，1个停止位）。冗长模式默认为启用，可通过注释`STL_VERBOSE_POR`和`STL_VERBOSE`（在B类配置文件`stm32xx_STLparam.h`中定义）在启动和/或运行期间禁用冗长模式。每次在运行时成功完成SRAM和Flash测试后，通过在终端窗口打印特定符号'#'或'*'而发出信号。软件包之间的冗长调试信息稍有不同。


 6 显示了冗长模式的示例。

图6. 冗长模式下STM32 demo超级终端输出窗口

```

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

***** Self Test Library Init *****
Start-up CPU Test OK
Pin reset
IWDG reset
WWDG reset
... WWDG reset, WDG test completed ...
Start-up FLASH 32-bit CRC OK
Control Flow Checkpoint 1 OK
Full RAM Test OK
Clock frequency OK
Control Flow Checkpoint 2 OK

STM32F2xx Cortex-M3
IEC60395 test @IARc
... main routine starts ...
#####*#####
#####*#####_

```

5.4.3 软件包调试

如果任何自检例程失败，则在定义 *stm32xx_STLstartup.c* 文件的 *FailSafePOR* 函数中触发 MCU 复位。这使应用调试变得困难，可引起调试器释放执行上下文。

在调试软件包时，一类方法是禁止某些功能：

- 复位微控制器时，禁用 *FailSafePOR()* 中的调用宏 *HAL_NVIC_SystemReset()*，以防止失去执行环境。可通过 *stm32xx_STLparam.h* 中注释 *NO_RESET_AT_FAIL_MODE* 的定义完成
- 增加或删除自检例程时，禁用控制流程监控，尤其是运行时自诊断。这可通过 *stm32xx_STLstartup.c* 文件中定义的函数 *control_flow_check_point()* 的重新定义来完成，
- 禁用所有程序内存 CRC 校验和测试，以防止在代码中使用软件断点时发生内存校验和错误。
- 禁用窗口看门狗，以防止出现程序的执行超出其刷新窗口（或保持其刷新窗口足够宽）。

或者是在调试阶段启用某些功能：

- 通过取消对***STL_VERBOSE_POR***或***STL_VERBOSE***的注释（在***stm32xx_STLparam.h***中），来启用冗长诊断模式，以便可以通过串口终端监视B类状态文本信息
- 启用指示测试阶段的LED输出信号，通过在***stm32xx_STLparam.h***中取消对***STL_EVAL_MODE***的注释完成。
- 通过在***stm32xx_STLparam.h***文件中包括***STL_EVAL_LCD flag***的定义，实现LCD控制。

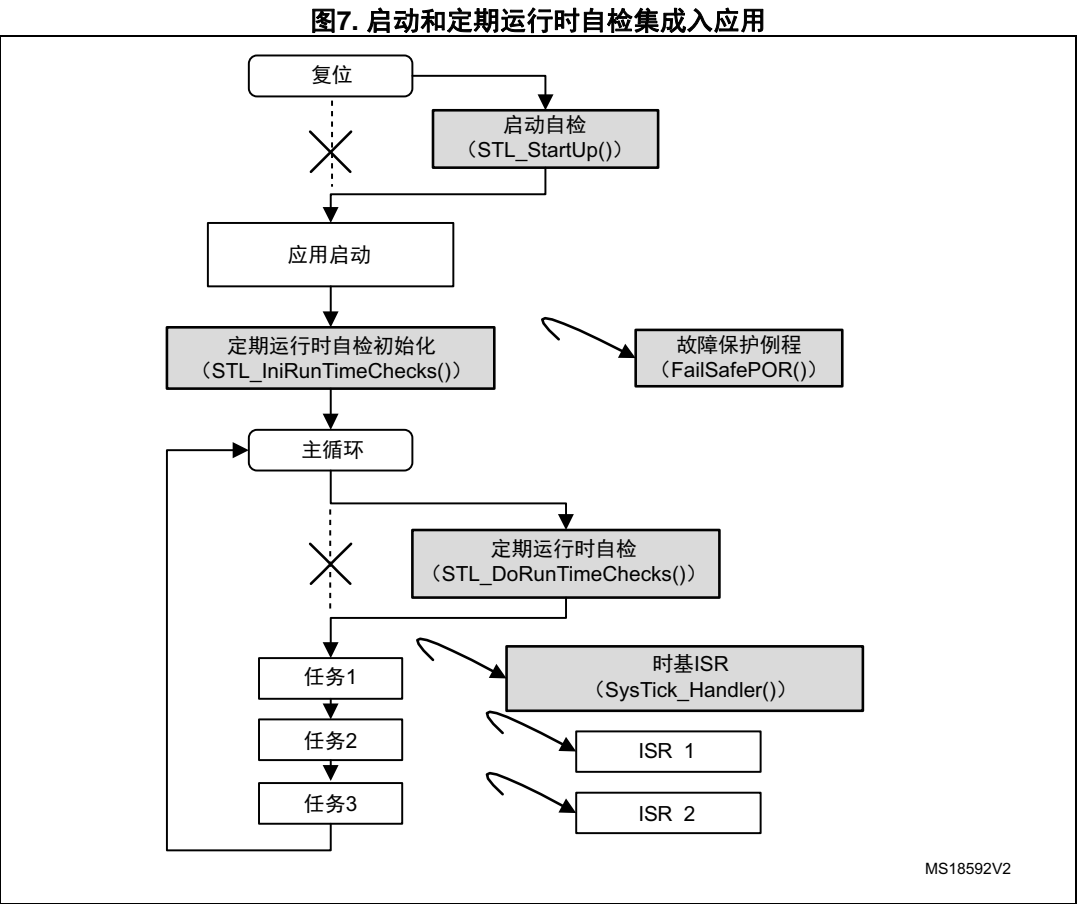
6 B类安全方案结构

6.1 与用户应用的集成

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

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

图 7显示B类软件包集成入用户软件解决方案的基本原理。



原则上，当将STL模块集成入应用时，用户必须提供以下步骤：

- 用户程序开始前执行初始启动测试
- 定期执行专门时间段（与安全时间相关）设置的运行时自检
- 应用运行时，设置独立和窗口看门狗，以防止其到期（理想的情况是通过STL测试周期对其刷新进行微调）
- 为FAM和Flash的启动和运行时测试设置正确的测试区
- 注意测试的错误结果，并进行合适的操作来保证系统的安全
- 处理安全状态程序及其内容
- 处理HardFault异常程序及其内容
- 防止可能与应用SW发生的冲突（尤其是中断、DMA、CRC - 见 表 13）
- 运行应用特定微控制器部件的测试（与应用安全任务相关）
- 排除所有与任何安全任务不相关的调试功能，仅将其用于调试或测试。

表13. STL进程可能与用户SW发生的冲突

测试	可能的冲突风险	源
CPU	被测试CPU寄存器的内容由用户SW更改	用户中断
RAM	被测试RAM的内容由用户SW更改	用户中断 或 DMA ACTIVITY
	在刚执行透明测试的块范围内，由用户完成的数据改变会被忽略或损坏	
Flash存储器	CRC计算损坏	CRC外设被一些其他用户组件使用 ⁽¹⁾
时钟	中断捕获服务延迟 (过捕获)	用户中断

1. 在这种情况下，用户必须处理CRC寄存器的内容，以保持测试的连续性。

在启动测试过程中启用任何调试支持时，用户必须确保通过适当的C编译器初始化，这是因为一些外设HAL驱动依赖复制进RAM的内容（如支持评估板固件的配置常量域）。由于该测试损坏了所有RAM内容，在完整的RAM测试完成后，这就成了一个问題。为了防止HAL驱动出现任何问题，如果RAM里的内容在RAM测试和进入main函数前的这段期间被使用到，用户必须确保恢复RAM内容。

当应用运行时，定期执行过程测试，间隔时间由B类变量 *TimeBaseFlag* 的值定义。这些重复的频率提供了基本安全间隔。其设置由用户定义，取决于应用限制和需求。当用户计算整个运行时间测试周期完成时，必须考虑RAM和Flash的测试区域及这些部分测试下的单个区域的大小。

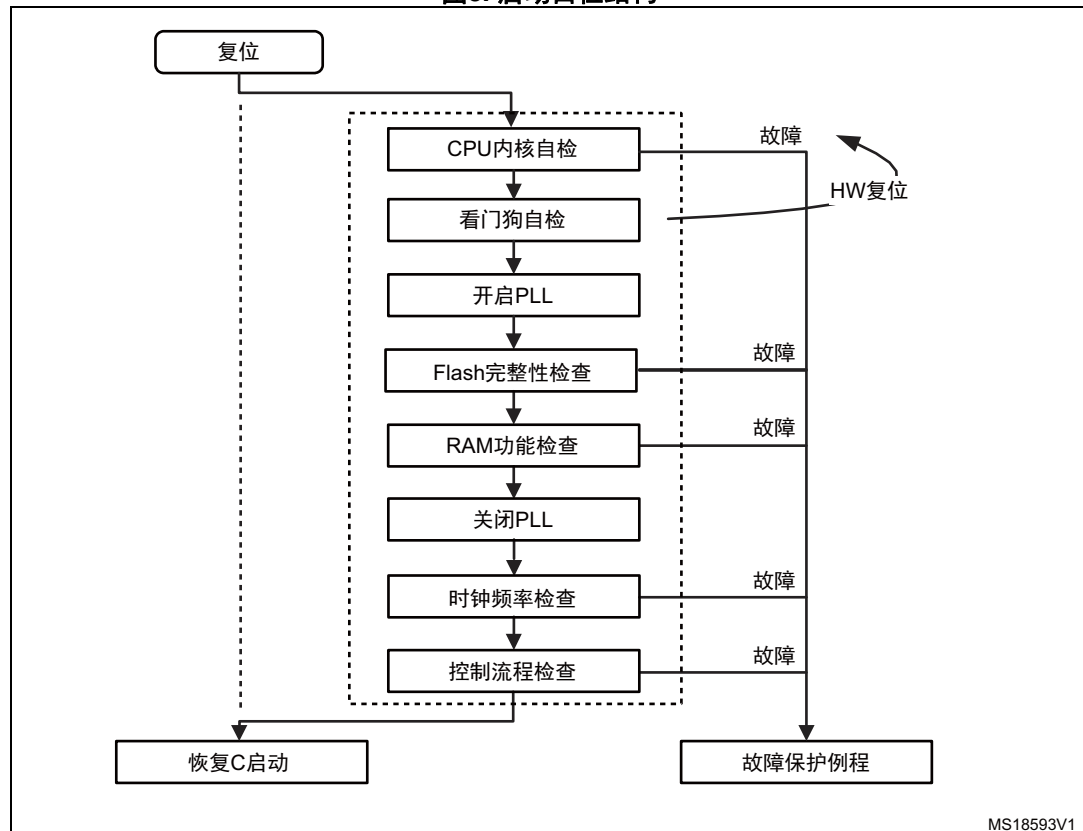
为初始化定期测试，用户必须在main函数开始时调用 *STL_InitRunTimeChecks()* 例程，然后确保main函数中 *STL_DoRunTimeChecks()* 的定期调用 - 在主循环内部是最佳情况。周期性来源于HAL层定义和初始化的SysTick中断。当定义了TimeBaseFlag后 *SysTick_Handler()* 中断服务计数1ms节拍并执行短期的局部透明RAM March C或March X检查（默认设为10 ms）以及主程序中其他的运行时测试。 *FailSafePOR()* 例程在 第 5.1.1 节 中讨论。

用户应特别注意，意外中断可能损坏检查进程（包括NMI，如果其使用目的不同于内部故障结果），应考虑此类事件的所有可能结果。用户还应确保没有中断能损坏STL测试执行或明显改变其定时。

6.2 启动自检说明

复位微控制器后，执行首次检查时，应在初始化阶段运行启动自检，如图8所示。

图8. 启动自检结构



启动测试结构如图9所示，包括以下自检：

- CPU启动测试
- 看门狗启动测试
- Flash完整校验和测试
- 完整RAM March C测试
- 时钟启动测试
- 控制流程检查

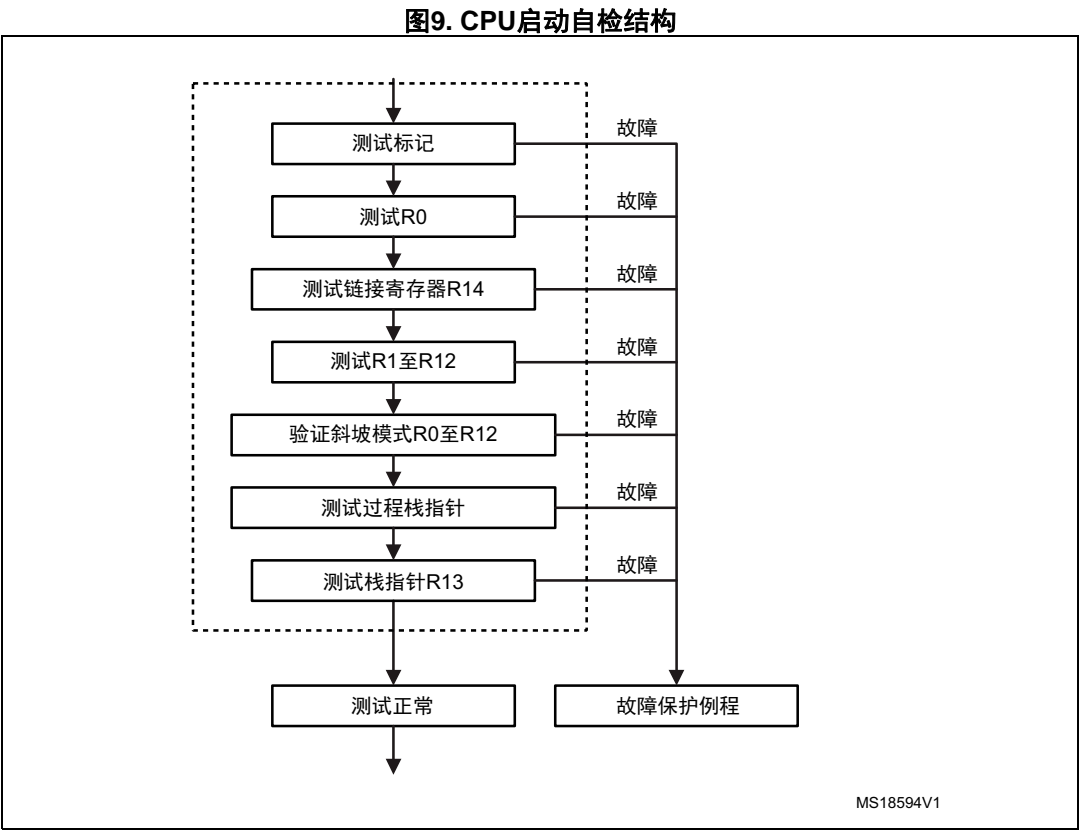
这些块在本节的剩余部分详细说明。

6.2.1 CPU启动自检

CPU启动自检检查内核标记、寄存器和栈指针的正确功能。如果发现任何错误，应调用故障安全例程。

源文件是汇编程序，因为不同的内核支持的指令集不同而稍有不同。IAR™、Keil® 和Ac6解决方案的汇编程序的版本不同。

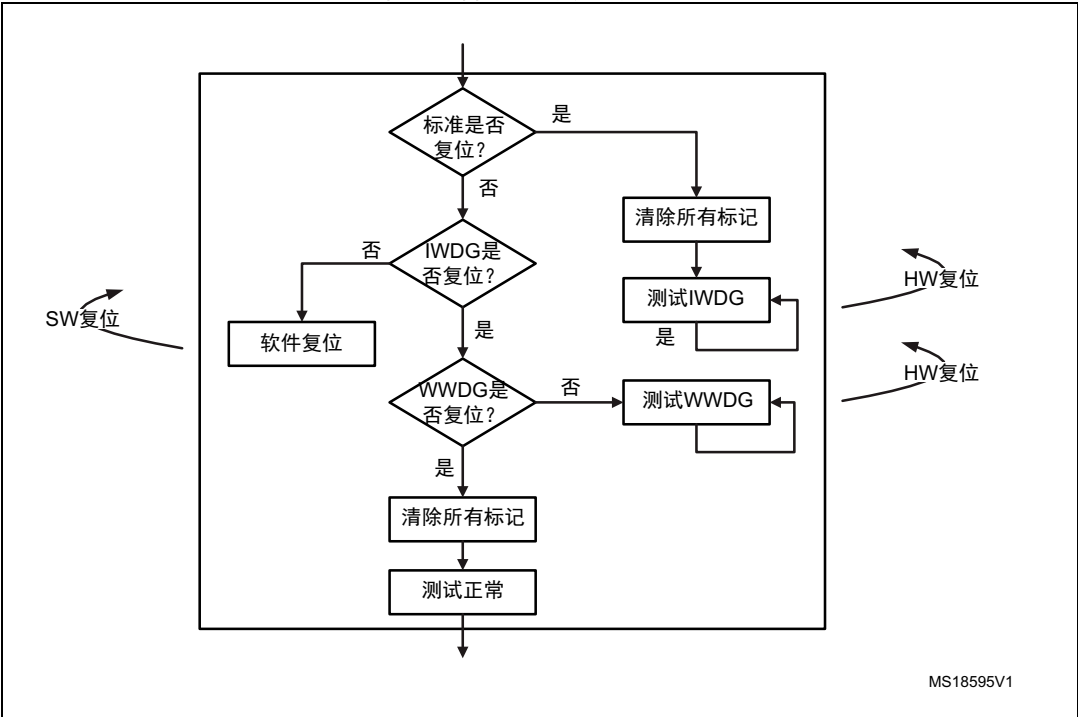
基本结构如 [图 9](#)所示。



6.2.2 看门狗启动自检

测试结构基于复位状态寄存器内容，通过相应标记寄存所有先前的复位原因，直至寄存器清零（见 [图 10](#)）。

图10. 看门狗启动自检结构



在看门狗测试开始时假设标准复位条件（接通电源、低功耗、软件或外部针标记发出之前复位原因的信号）。IWDG设置为最短复位周期，所有复位标志被清除，当时间到了以后，IWDG应该触发复位。在下一次复位后，IWDG标记应被设置，并且是唯一的复位原因。接下来继续WWDG的测试。当复位状态寄存器内的两个标记都置位时，测试被认为完整，且复位状态寄存器内的所有标记都被再次清除。

用户必须注意IWDG和WWDG的正确设置。其刷新窗口的周期和参数必须根据时基间隔而设置，这是因为正常刷新在成功完成主循环的周期运行时测试时执行。

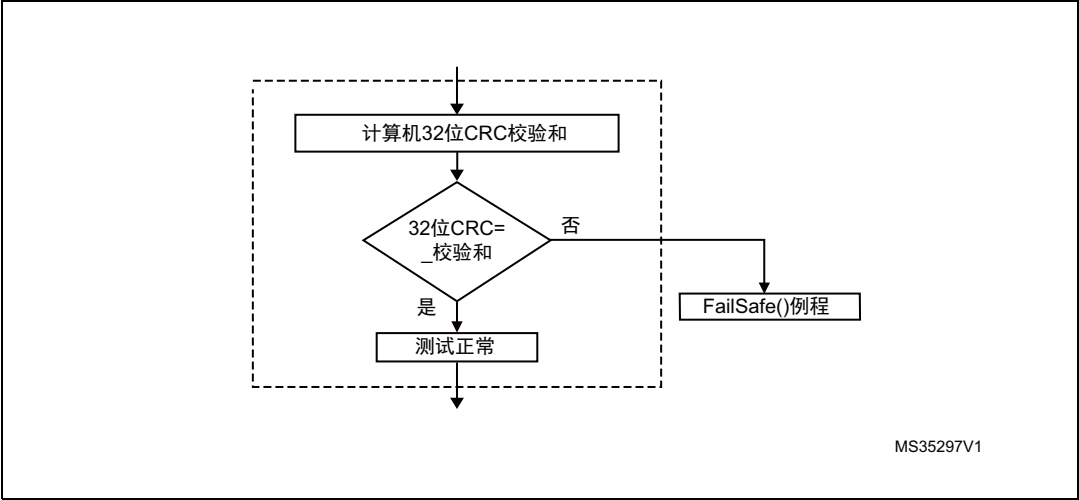
系统节拍中断服务例程通过专用时基标记指示确定的时基间隔。主程序中的运行时测试按这个间隔循环进行。由于看门狗控制是成功完成运行时测试的最后一步（应为主循环内唯一刷新看门狗的位置），时基间隔的设置必须与看门狗超时相关，反之亦然。

看门狗刷新周期必须短于看门狗超时，以防止CPU复位，如 [图 15](#)所示。

6.2.3 Flash存储器完整校验和自检

CRC校验和计算在整个Flash存储器空间（在链接器定义）执行。计算结果与链接器中的值相比较：如果不相同，则测试失败（见[图 11](#)）。

图11. Flash启动自检结构



关于CRC程序的附加注释，参考 [第 5.4.3节](#)。

6.2.4 完整RAM March-C自检

在如 [图 12](#)所示的6个循环中，交替检查整个RAM空间，并用背景模式（值0x00000000）和反向背景模式（值0xFFFFFFFF）逐字填充。前三个循环按地址的递增顺序执行，后三个循环按相反的顺序执行。

可对一些产品的测试地址顺序进行加扰，因为其遵循物理地址顺序，从而防止并识别相邻物理存储单元之间的任何串音。加扰原理如 [表 14](#)所示。

基本物理单元是一个覆盖16字块的模式（一行）。单元格内的编号代表逻辑地址，而其他顺序则代表物理布局。粗框架强调逻辑顺序被加扰的位置。

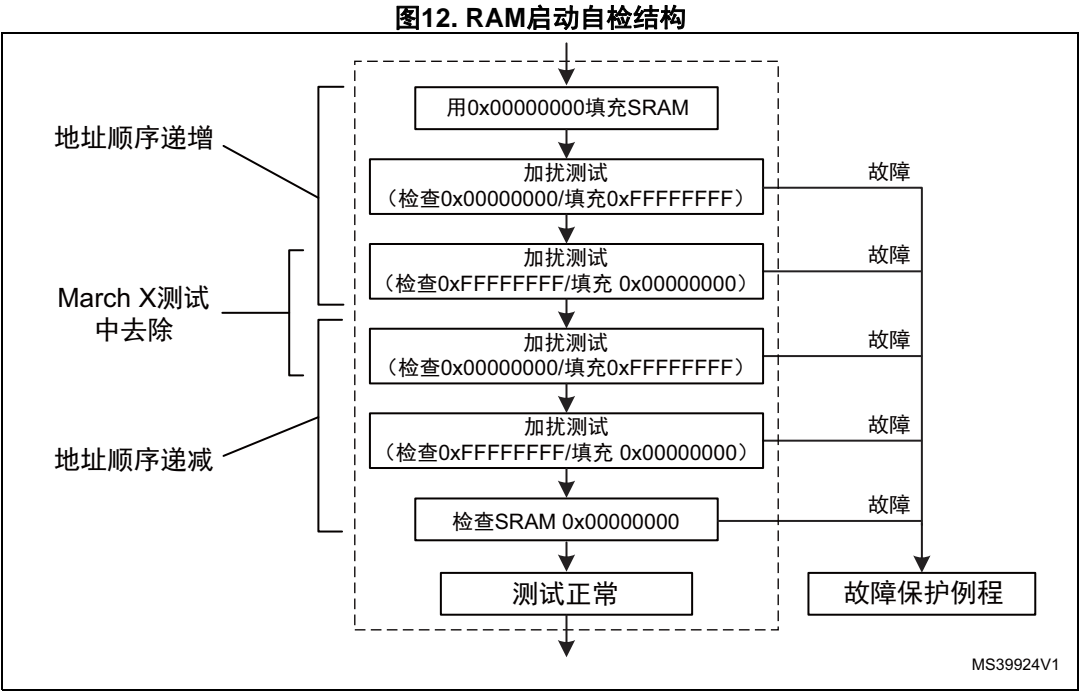
表14. RAM物理地址顺序被组织成16字块

	物理地址顺序---->															
行-->	0	1	3	2	4	5	7	6	8	9	11	10	12	13	15	14
	16	17	19	18	20	21	23	22	24	25	27	26	28	29	31	30
	32	33	35	34	36	37	39	38	40	...						

新ST产品没有地址加扰。对于那些执行加扰的产品，用户必须使用ARTISAN汇编编译参数（[第 3.3节](#)中显示更多详情）。

一些新产品已执行硬件写保护，单位冗余（硬件奇偶校验）应用于CCMRAM或部分SRAM。这一方面在 [第 3节](#)中讨论。

整个测试循环的算法如 [图 12](#)所示。如果检测到错误，则测试中断，返回故障结果。



注：RAM测试主要面向字，但可采用更精确的位推进测试算法。然而，该方法被认为更耗费时间和代码。

6.2.5 时钟启动自检

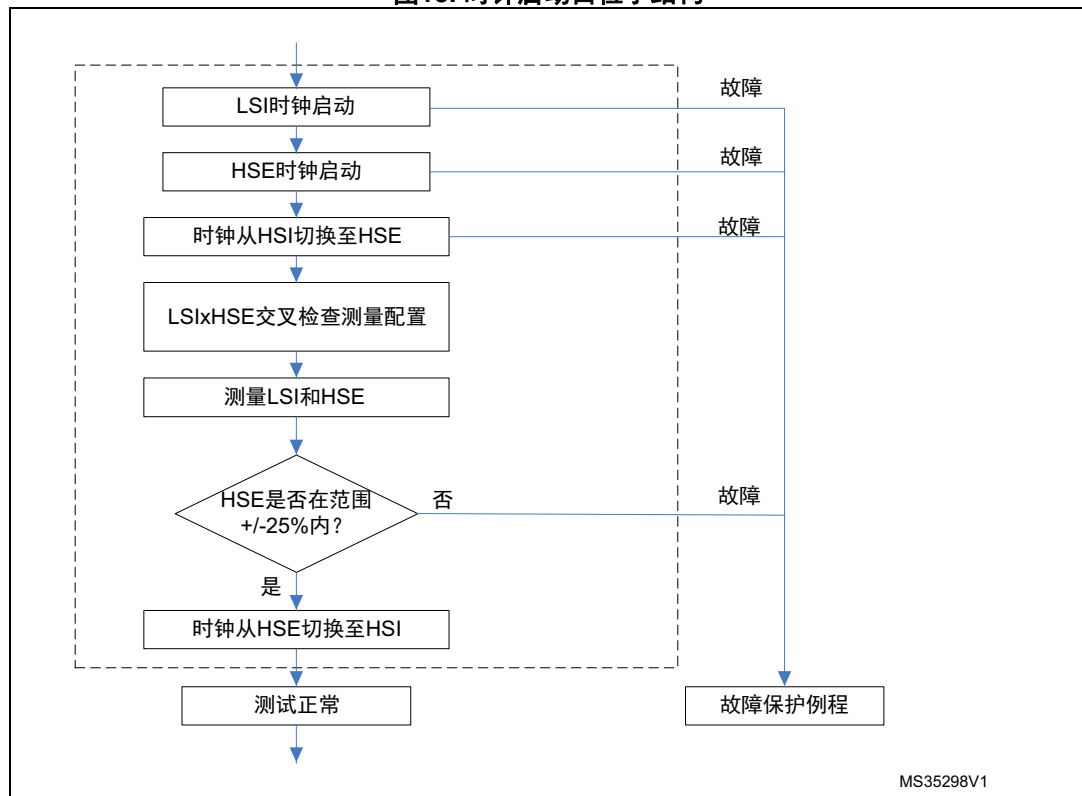
测试流程如 [图 13](#)所示。

首先，内部低速时钟（LSI）源启动。下一步，外部高速时钟源（HSE）启动，由HSE作为时钟源的PLL设为系统时钟。专用计时器初始化，用于HSE频率的交叉参考测量，由预先定义的LSI周期数控制。此类测量专用的定时器和通道跟产品相关^(a)。该定时器计数器连续两次获取的值不同，从而可得出LSI和HSE频率之间的比率。获取的值及其差别在定时器信道的定期中断服务期间处理。将比率测量与预期范围相比较，如果与标称值的差超过 $\pm 25\%$ ，则报错。范围由宏 **HSE_LimitHigh()**和**HSE_LimitLow()**根据一些常量定义在 **stm32xx_STLparam.h**文件中定义。

用户负责正确输入这些宏，并考虑这些限值是否能保持恒定，或这些限值是否根据应用而动态调整（例如，由于整个温度范围内参考时钟信号的准确性）。测试完成后，CPU时钟切换回默认HSI源。

a. 处理时钟交叉检查初始配置的产品特定函数 **STL_InitClock_Xcross_Measurement()**并不属于保存通用STL代码的文件，而是在文件 **stm32yyxx_it.c**中执行，该文件里包含了所有应用于设备的相关中断服务。

图13. 时钟启动自检子结构



6.2.6 控制流程检查

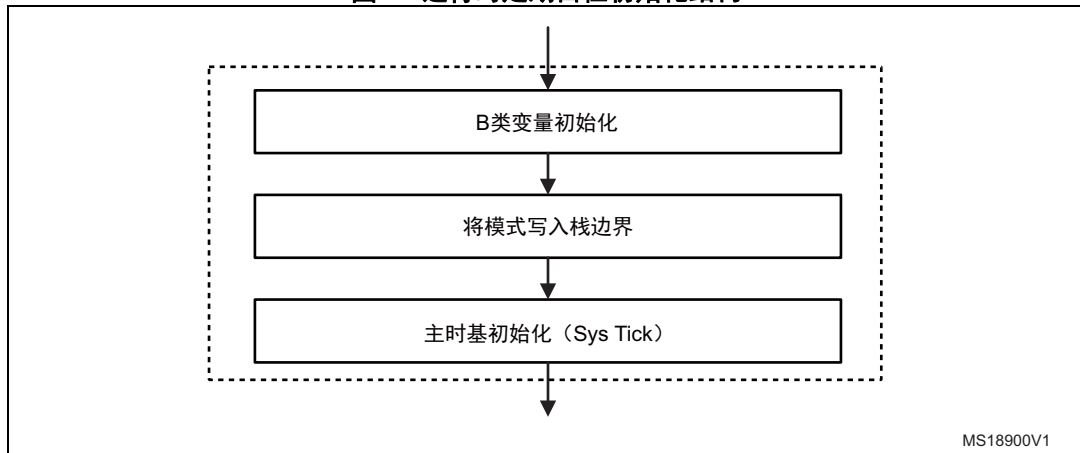
启动测试通过控制流程检查点程序完成。在完成前，一个特殊数据组合储存在堆栈的顶部。

6.3 定期运行时自检初始化

假设所有启动自检都成功通过且已完成标准的初始化，则运行时自检包必须在程序进入主循环前进行初始化，同时执行运行时自检的定期调用（见 [图 14](#)）。应正确设置定时，以确保运行时测试程序在必要的间隔被调用，从而保持足够的应用过程安全时间（关于更多详情，见 [第 5.3 节：执行定时测量和控制](#)）。

所有B类变量都会被初始化。零及其互补值储存在每个B类变量互补对中。为系统时钟和参考频率交叉检查测量配置专用定时器。使用相同的启动测试方法。

图14. 运行时定期自检初始化结构



6.4 运行时定期自检说明

6.4.1 运行时自检结构

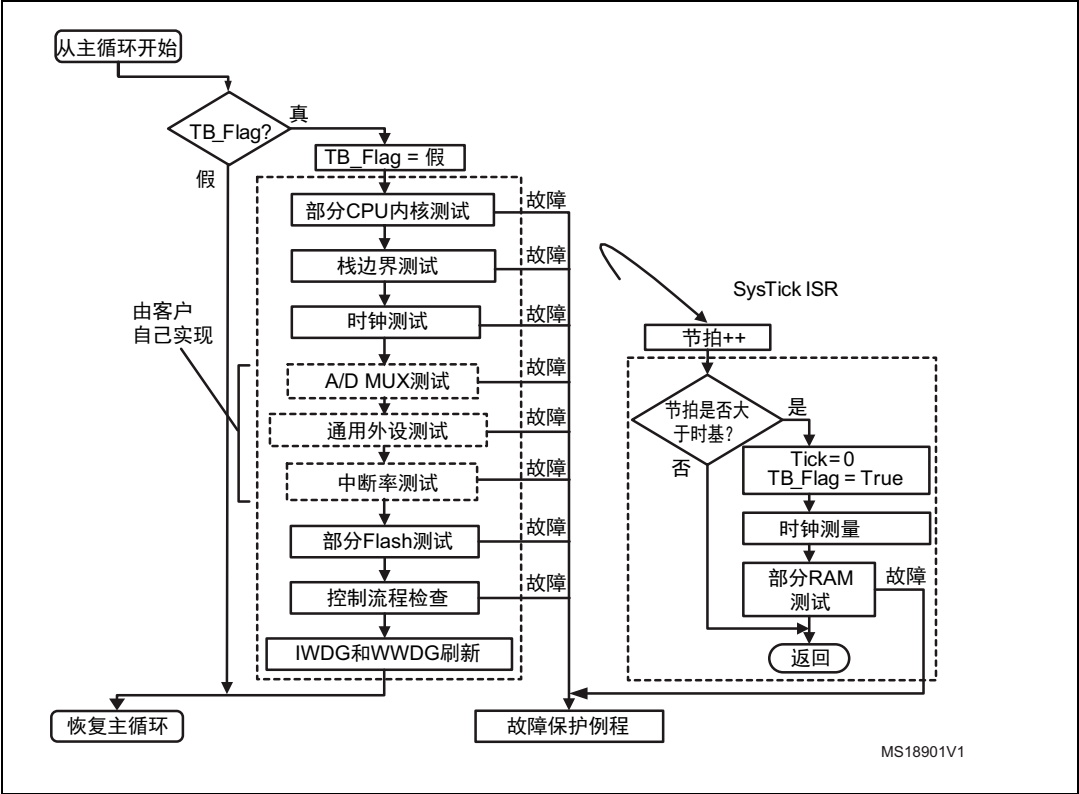
运行时测试是在主循环定期执行的测试块。执行时间取决于时基中断设置。首次运行前，包含的所有测试必须在运行时初始化阶段进行初始化（参考图 7）。

当用户从主循环调用其执行时，此处的大部分测试都根据TimeBaseFlag信号定期执行。仅部分透明RAM测试和时钟测量备份在SysTick和专用定时器中断服务内执行。

运行时应包括以下列出的测试：

- CPU内核部分运行时测试
- 栈边界上溢测试
- 时钟运行时测试
- AD MUX自检（未执行）
- 中断频率测试（未执行）
- 通信外设测试（未执行）
- Flash部分CRC测试（包括完整测试评估）
- 独立和窗口看门狗
- 部分透明RAM March C/X测试（系统中断范围）。

图15. 定期运行时自检和时基中断服务结构

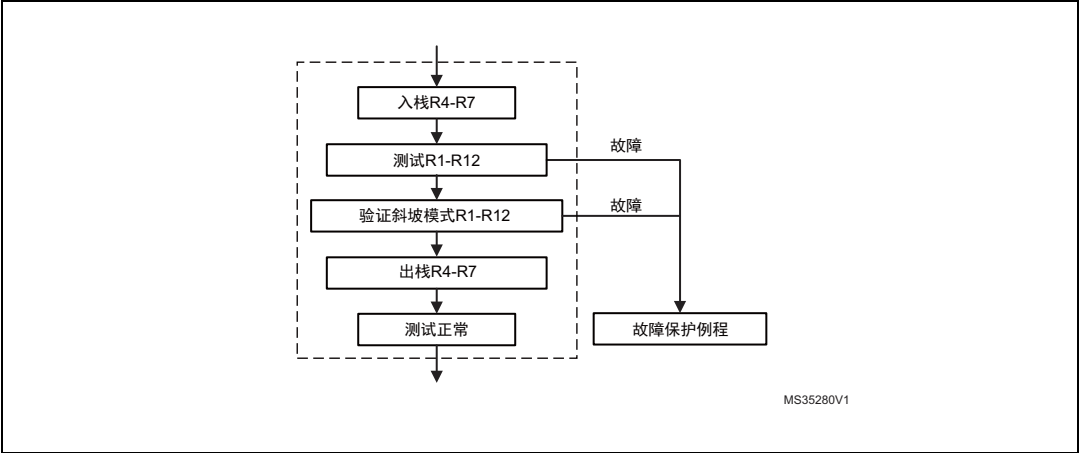


注：不包括模拟部件、通信外设和应用中断发生率测试，其实现取决于设备能力和用户的应用需求。

6.4.2 简化版CPU运行时自检

运行时CPU内核自检是简化版的运行时测试，如第 6.2.1 节中说明。此处不测试标记和栈指针。

图16. CPU灯运行时自检结构



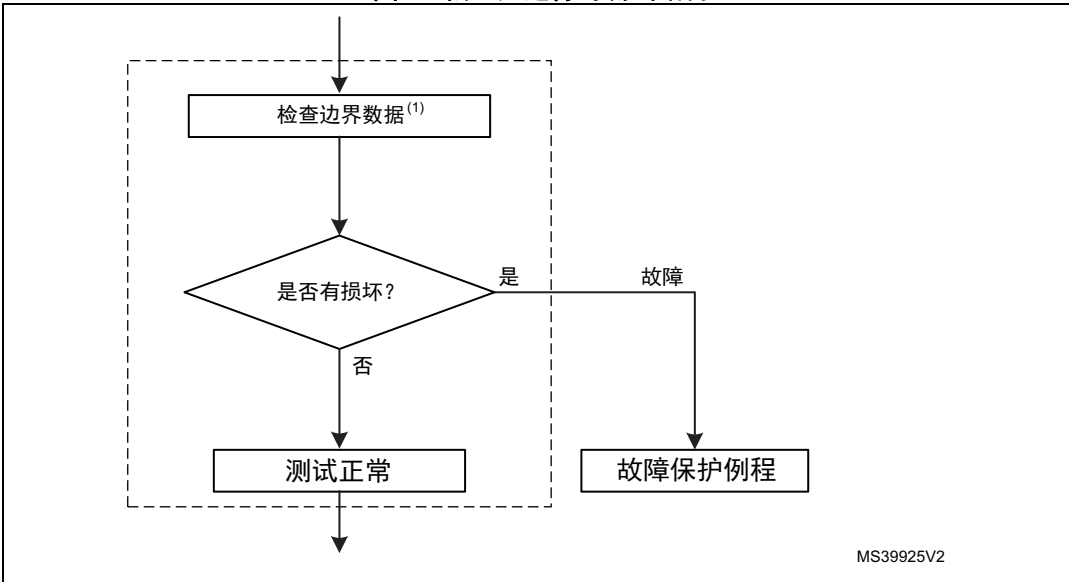
如果任何错误代码返回，则调用故障保护程序。

6.4.3 栈边界运行时测试

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

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

图17. 栈上溢运行时测试结构



1. 如果栈区域不在RAM空间的实际终端，必须通过检查顶部特殊数据组和完整性来检查是否有栈下溢情况。

6.4.4 时钟运行时自检

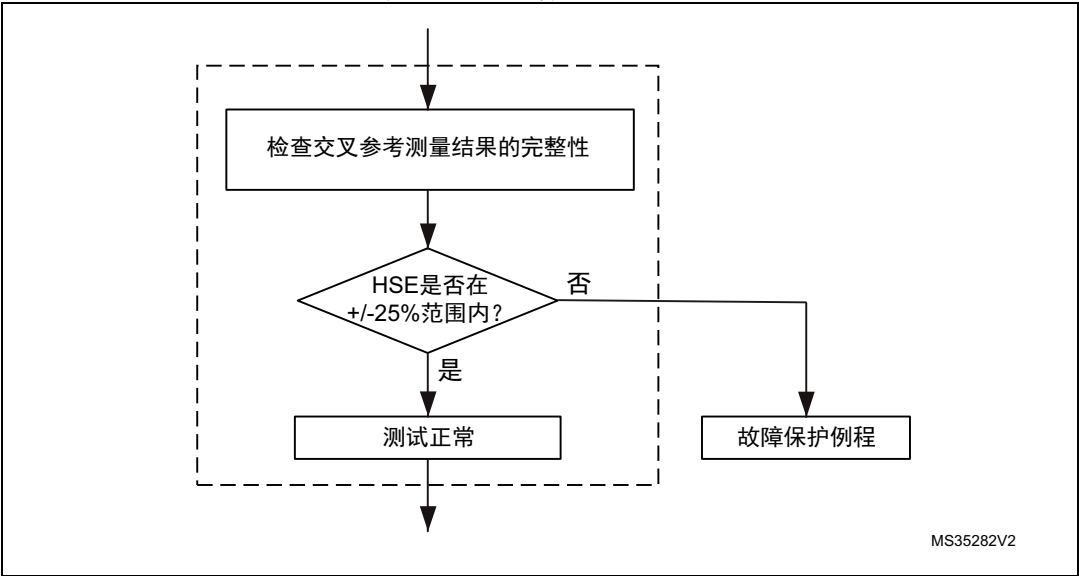
时钟运行时自检采用与启动自检相似的程序（见第 6.2.5 节）。时钟交叉参考率的真实性检查取决于定时器捕获事件最后两次结果的差别。在专用定时器定期中断服务过程中，对这些结果加以储存，并提供系统（HSE）和参考（LSI）频率间的交叉参考测量。

测试检查HSE率是否在预期范围内（标称值的+/- 25%）。如果发现差别较大，或HSE信号丢失，或测量中断消失，则CPU时钟源会立即切换回HSI且HSE故障状态返回。否则，测试返回至正常状态。

在对HSE范围比较前，会检查所有和报告时钟检测结果有关的变量的完整性。

如果使用HSE，应对另一个应用时钟源（如HSI）进行交叉检查。用户必须修改专用定时器的设置，以确保正确的时钟交叉测量（见stm32xxx_it.c文件中定义的STL_InitClock_Xcross_Measurement()函数）并检查这种测量是否受实际产品支持。

图18. 时钟运行时自检结构



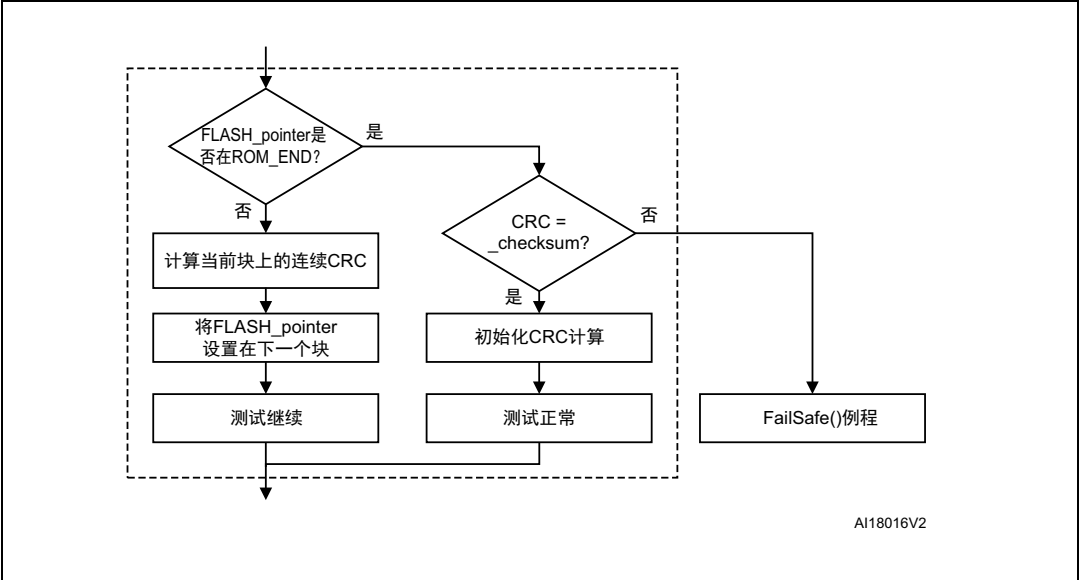
6.4.5 局部Flash CRC运行时自检

使用内置CRC HW块时，每步都要执行Flash块部分32位CRC校验和。用户必须正确定义总测试区域及每个测试步骤所涉及的块大小。当执行完最后一块后，将CRC校验和和链接器内储存的值进行对比。如果出现差别，则调用故障保护例程，否则对新计算周期进行初始化。

该测试在对一个Flash块进行计算前检查所有相关变量的完整性。

关于CRC程序的附加注释，参考 [第 5.4.3节](#)和 [第 3.4节](#)。

图19. 局部Flash CRC运行时自检结构



6.4.6 运行时测试中的看门狗服务

如果运行时服务模块成功完成，则窗口和独立看门狗在返回主循环前的最后一步刷新。对于需适当的定时调入运行时模块，这一点至关重要。模块应调用的周期由在 **STL_DoRunTimeChecks()** 例程开始时测试的时基标记发出内部信号（见 [图 15](#)）。用户必须确保不在传送该程序的调用，从而能针对时基标记变化做出反应，并在正确的间隔刷新看门狗。

为有效使用看门狗，只应在主循环上的一个位置进行刷新，这一点很重要。不应有其他看门狗刷新，除 **STL_DoRunTimeChecks()** 例程中的刷新外。有时，还需要在流程的初始阶段刷新看门狗。在这种情况下，刷新不应在任何软件的无限循环内。理想情况下，应仅放在编码的简单部分。

6.4.7 部分RAM运行时自检

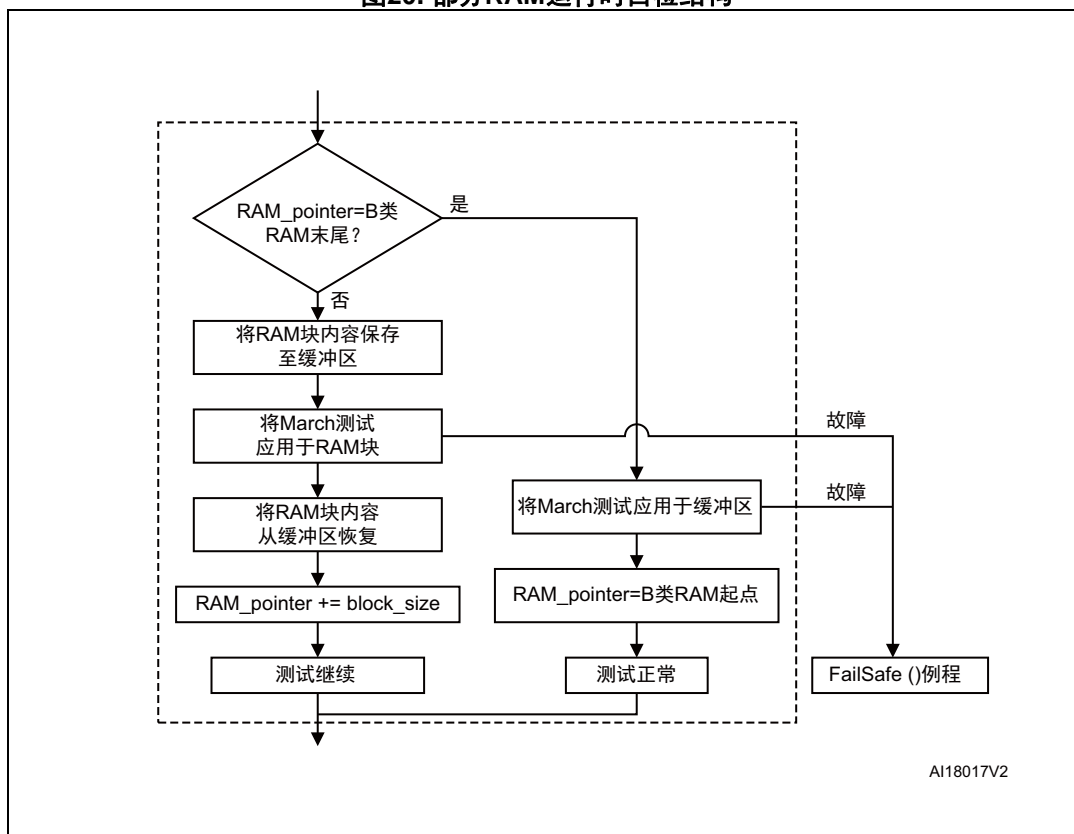
在时基中断服务例程内部逐步执行局部RAM测试。该测试包括指派给B类变量的RAM部分。默认情况下，每一步都要测试含有六个字的区域。为保障耦合故障覆盖率，每个测试的内存块都始终与测试上一步和下一步的两个附加的相邻字重叠。

对于有SRAM奇偶校验HW检查的设备，该测试可跳过，如 [第 3.3节: SRAM测试](#)所示。

[表 15](#)中总结了该块的测试运行，注意必须始终遵循内存中地址的物理顺序，无论测试执行是升序填充（检查）还是降序填充（检查）。如果内存单元的物理顺序不连续，则用户必须将块大小保持在多重字地址，与应用加扰模式的重复相应。如果不应用加扰，可测试该块内任意后续字的编号。若需有关加扰的更多详细信息，请参见 [第 6.2.4节](#)。

该测试在进行块检测前检查所有相关变量的完整性。

图20. 部分RAM运行时自检结构



分三个阶段执行单个测试步骤，如图 21和图 22所示：第一个表示连续地址模型，第二个可参考物理加扰模型（灰色方框强调不同的非连续测试顺序）。

在第一阶段，块测试（单元D1至D4）涉及的所有单元和重叠字（单元D0至D5）的当前内容储存在存储器缓冲区。下一阶段，对测试RAM块内的所有这些字执行破坏性的March测试。在最后阶段，存储缓冲区的初始内容返回至测试位置。

在整个序列的最后一步，存储缓冲区自身通过March测试进行测试。缓冲区与下两个附加的相邻字一起测试，以覆盖缓冲区自身的耦合故障。存储缓冲区的大小必须适应测试块的大小。存储缓冲区成功测试后，整个测试被重新初始化，然后重新开始。如果检测到任何故障，则调用故障保护例程。该测试在块检测前检查所有相关变量的完整性。

用于部分测试的块大小默认为四个字（下两个涉及重叠测试）。该模型相当于加扰模式的重复。当用户更改该块的大小时，算法必须遵循加扰时间（如存在）。如果设计SRAM时没有加扰，则块大小不受约束，但是用户必须修改写入汇编程序的例程（针对固定默认块大小而写入）。

图21. 部分RAM运行时自检 - 故障耦合原理（无加扰）

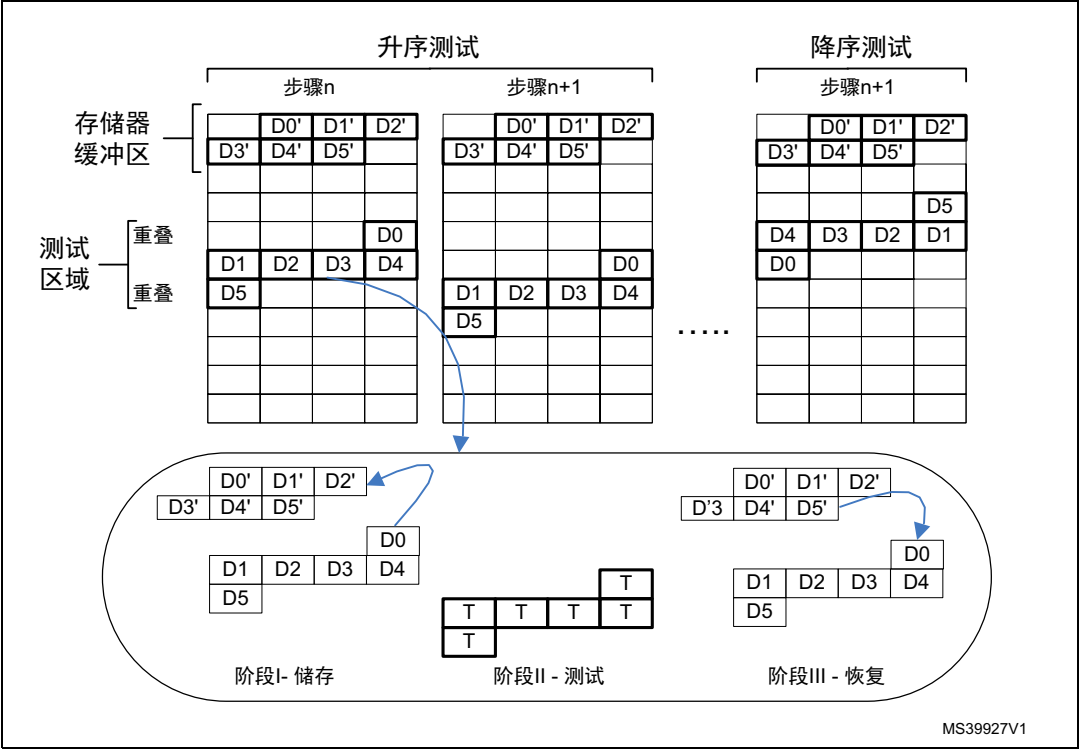
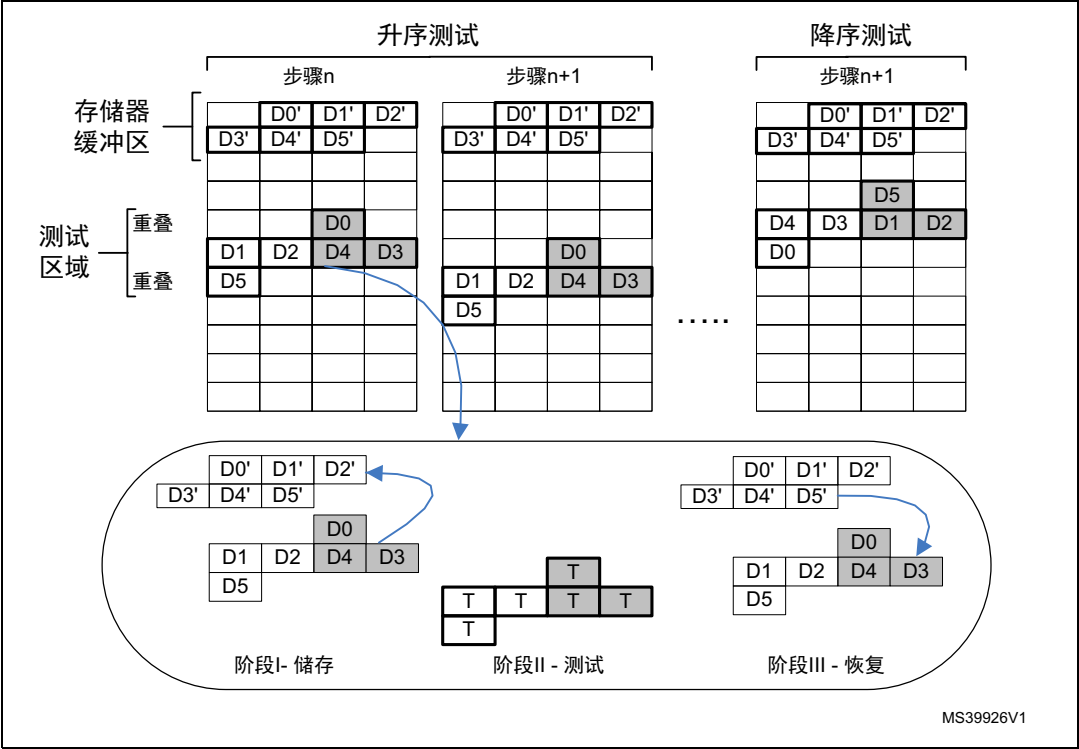


图22. 部分RAM运行时自检 - 故障耦合原理（有加扰）



注：应遵循地址加扰顺序，见表 14。

操作顺序在 表 15中给出。当定义了USE_MARCHX_TEST时，可用March X算法取代March C。由于跳过了两个中间的推进步骤（第2步和第3步），该测试速度较快。

表15. RAM部分测试的March C阶段

March阶段	块上的部分字测试	地址顺序
初始	写0x00000000模式	递增
1	检查0x00000000模式， 写0xFFFFFFFF模式	递增
2	检查0xFFFFFFFF模式， 写入0x00000000模式	递增
3	检查0x00000000模式， 写0xFFFFFFFF模式	递减
4	检查0xFFFFFFFF模式， 写入0x00000000模式	递减
5	检查0x00000000模式	递减

7 版本历史

表16. 版本历史

日期	版本	变化说明
2014年6月20日	1	初始版本
2016年1月12日	2	<p>更改了文件分类。</p> <p>更新了前言, 第 1 节: 参考文档, 第 2 节: 软件包变化概述, 第 3 节: STL 软件包之间的主要差别 (从产品的角度来看), 第 3.3 节: SRAM 测试, 第 3.4 节: Flash 存储器完整性测试, 第 3.6 节: 固件配置参数, 第 3.7 节: 固件集成, 第 3.8 节: HAL 驱动接口, 第 4 节: 符合 IEC, UL 和 CSA 标准, 第 4.2 节: 应用相关特定测试不在 ST 固件自检库范围内, 第 4.3 节: 安全生命周期, 第 5.1.1 节: 故障安全模式, 第 5.1.2 节: 安全相关变量和栈边界控制, 第 5.1.3 节: 流程控制程序, 第 5.2.1 节: 软件包所含项目, 第 5.2.3 节: 定义新的安全变量和受检内存区, 第 5.2.4 节: 应用实现示例, 第 5.4.1 节: 配置控制, 第 6.2.4 节: 完整 RAM March-C 自检, 第 6.4.4 节: 时钟运行时自检以及第 6.4.7 节: 部分 RAM 运行时自检。</p> <p>更新了表 1: STL 软件包概述, 表 2: FW 结构的主要组织, 表 3: 通用 STL 软件包的结构, 表 4: 产品特定 STL 软件包结构, 表 8: 所有 STL 栈程序使用的 HAL 驱动概述以及表 15: RAM 部分测试的 March C 阶段。</p> <p>更新了图 1: RAM 存储器配置示例, 图 2: 控制流程四步检查原理, 图 12: RAM 启动自检结构, 图 15: 定期运行时自检和时基中断服务结构, 图 17: 栈上溢运行时测试结构以及图 22: 部分 RAM 运行时自检 - 故障耦合原理 (有干扰)。</p> <p>增加了第 5.3 节: 执行定时测量和控制。</p> <p>增加了表 6: 不同 STM32 微控制器之间的兼容性, 表 7: 如何管理兼容性 & 配置 STL 软件包, 表 11: 用于定时测量的信号以及表 12: 结果对比。</p> <p>增加了图 4: 启动期间典型的测试时序和图 5: 运行期间典型的测试时序。向图 17 添加了脚注 1。</p>
2016年3月7日	3	<p>更改了文件分类。</p> <p>更新了前言。</p> <p>更新了表 6: 不同 STM32 微控制器之间的兼容性。</p>
2017年1月26日	4	<p>更新了文档标题和前言。</p> <p>更新了第 2 节: 软件包变化概述, 第 3.2 节: 时钟测试和时基间隔测量, 第 3.3 节: SRAM 测试, 第 3.4 节: Flash 存储器完整性测试, 第 3.5 节: 启动和系统初始化, 第 4 节: 符合 IEC, UL 和 CSA 标准, 第 4.2 节: 应用相关特定测试不在 ST 固件自检库范围内, 编码, 第 5.1.2 节: 安全相关变量和栈边界控制, 第 5.1.3 节: 流程控制程序, 第 5.2.1 节: 软件包所含项目, 第 5.2.3 节: 定义新的安全变量和受检内存区, 第 5.3 节: 执行定时测量和控制, 第 5.4.1 节: 配置控制, 第 5.4.3 节: 软件包调试, 第 6.2.5 节: 时钟启动自检, 第 6.3 节: 定期运行时自检初始化以及第 6.4.1 节: 运行时自检结构。</p> <p>更新了表 1: STL 软件包概述、表 4: 产品特定 STL 软件包结构、表 5: 集成支持文件、表 6: 不同 STM32 微控制器之间的兼容性和表 12: 结果对比。</p> <p>更新了图 5: 运行期间典型的测试时序。</p>

表17. 中文版本历史

日期	版本	变化说明
2018年9月3日	1	中文初始版本

重要通知 - 请仔细阅读

意法半导体公司及其子公司（“ST”）保留随时对 ST 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。买方在订货之前应获取关于 ST 产品的最新信息。ST 产品的销售依照订单确认时的相关 ST 销售条款。

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

ST 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 ST 产品如有不同于此处提供的信息的规定，将导致 ST 针对该产品授予的任何保证失效。

ST 和 ST 徽标是 ST 的商标。所有其他产品或服务名称均为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。本文档的中文版本为英文版本的翻译件，仅供参考之用；若中文版本与英文版本有任何冲突或不一致，则以英文版本为准。

© 2018 STMicroelectronics - 保留所有权利