
N32G003系列Class B应用笔记

简介

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。

本文档介绍了该工程在 N32G003 的 MCU 中如何执行 IEC 60730 所要求的软件安全性相关的操作及相关应用代码内容。

本文档适用于国民技术的 N32G003 系列产品。

目录

目录	2
1. IEC60730 class B 软件标准介绍.....	1
2. Class B 软件包说明.....	1
3. 测试点流程说明	2
3.1 启动时检测流程.....	3
3.1.1 CPU 启动时检测.....	4
3.1.2 看门狗启动时检测.....	5
3.1.3 FLASH 启动时检测	6
3.1.4 RAM 启动时检测	9
3.1.5 时钟启动时检测.....	10
3.1.6 控制流启动时检测.....	11
3.2 运行时检测流程.....	11
3.2.1 CPU 运行时检测.....	12
3.2.2 堆栈边界运行时溢出检测.....	13
3.2.3 系统时钟运行时检测.....	14
3.2.4 FLASH 运行时检测	15
3.2.5 看门狗运行时检测.....	16
3.2.6 局部 RAM 运行时自检	16
4. 软件库移植要点	19
5. 历史版本	20
6. 声明	21

1. IEC60730 class B 软件标准介绍

为确保电器使用安全，需要对软件运行时的风险控制措施进行评估。

国际电工委员会颁布的 IEC60730 引入了家用电器软件评估要求，附录 H(H.2.21)中对软件进行了分类：

A 类软件：软件仅实现产品的功能，不涉及产品的安全控制。比如室用恒温器的软件，灯光控制的软件...

B 类软件：软件的设计要防止电子设备的不安全操作。例如带自动门锁控制的洗衣机软件，带过热控制的电磁炉软件...

C 类软件：软件的设计为了避免某些特殊的危险。例如自动燃烧器控制和封闭的热水器的热切断（主要针对一些会引起爆炸的设备）

其中 B 类软件的具体评估要求，包含了需要检测的组件及相关故障和测试方案，整理见下表（参考 IEC60730 表 H.11.12.7）：

需要检测的组件		故障/错误	故障分类	nations 提供库	测试方案概述
1.CPU	1.1 寄存器	滞位(Stuck at)	MCU 相关	是	写相关寄存器并检查
	1.3 程序计数器	滞位(Stuck at)	MCU 相关	是	PC 跑飞则启动看门狗复位
2.中断		没有中断或者中断太频繁	应用相关	否	计算进中断次数
3.时钟		错误的频率	MCU 相关	是	使用 HSI 测 LSI 时钟频率
4.存储器	4.1 非易失存储器	所有的单比特错误	MCU 相关	是	FLASH CRC 完整性校验
	4.2 易失存储器	DC fault	MCU 相关	是	1. SRAM March C测试 2. 堆栈溢出检测
	4.3 寻址（与非易失和易失存储器相关）	滞位(Stuck at)	MCU 相关	是	FLASH/SRAM 测试已包含
5.内部数据路径	5.1 数据	滞位(Stuck at)	MCU 相关	否	仅针对使用外部存储器的 MCU，单片 MCU 不要求
	5.2 寻址	错误的地址	MCU 相关	否	
6.外部通信	6.1 数据	汉明距离 3	应用相关	否	数据传输中增加校验
	6.2 寻址	错误的地址	应用相关	否	
	6.3 时序	错误的时序	应用相关	否	计算通信事件的次数
7.输入输出	7.1 数字 I/O	H27 中定义的错误	应用相关	否	无
	7.2 模拟输入输出	H27 中定义的错误	应用相关	否	无

2. Class B 软件包说明

该应用笔记 Class B 软件包包含 CPU、时钟、Flash、RAM 等关键组件自检，代码部分随 SDK 发布，工程目录如下（以 Nations.N32G003_Library.1.2.0 版本为例）：



名称	修改日期	类型
ClassB_demo	2026/3/6 16:42	文件夹

3. 测试点流程说明

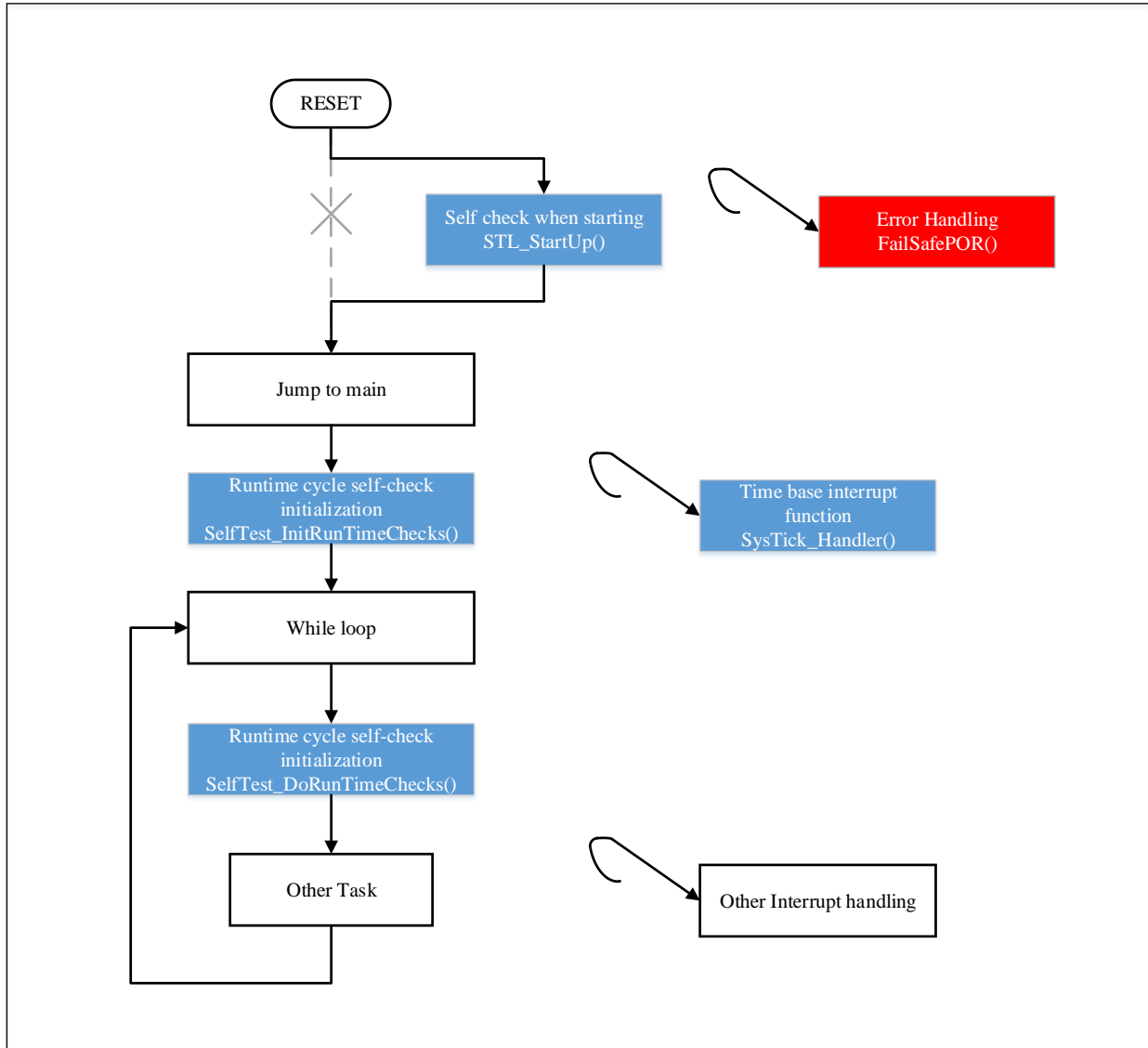
Class B软件包程序检测内容分为两个主要部分：启动时的自检和运行时的周期自检。
启动时的自检包括：

- CPU检测
- 看门狗检测
- Flash完整性检测
- RAM功能检测
- 系统时钟检测
- 控制流检测

运行时的周期自检：

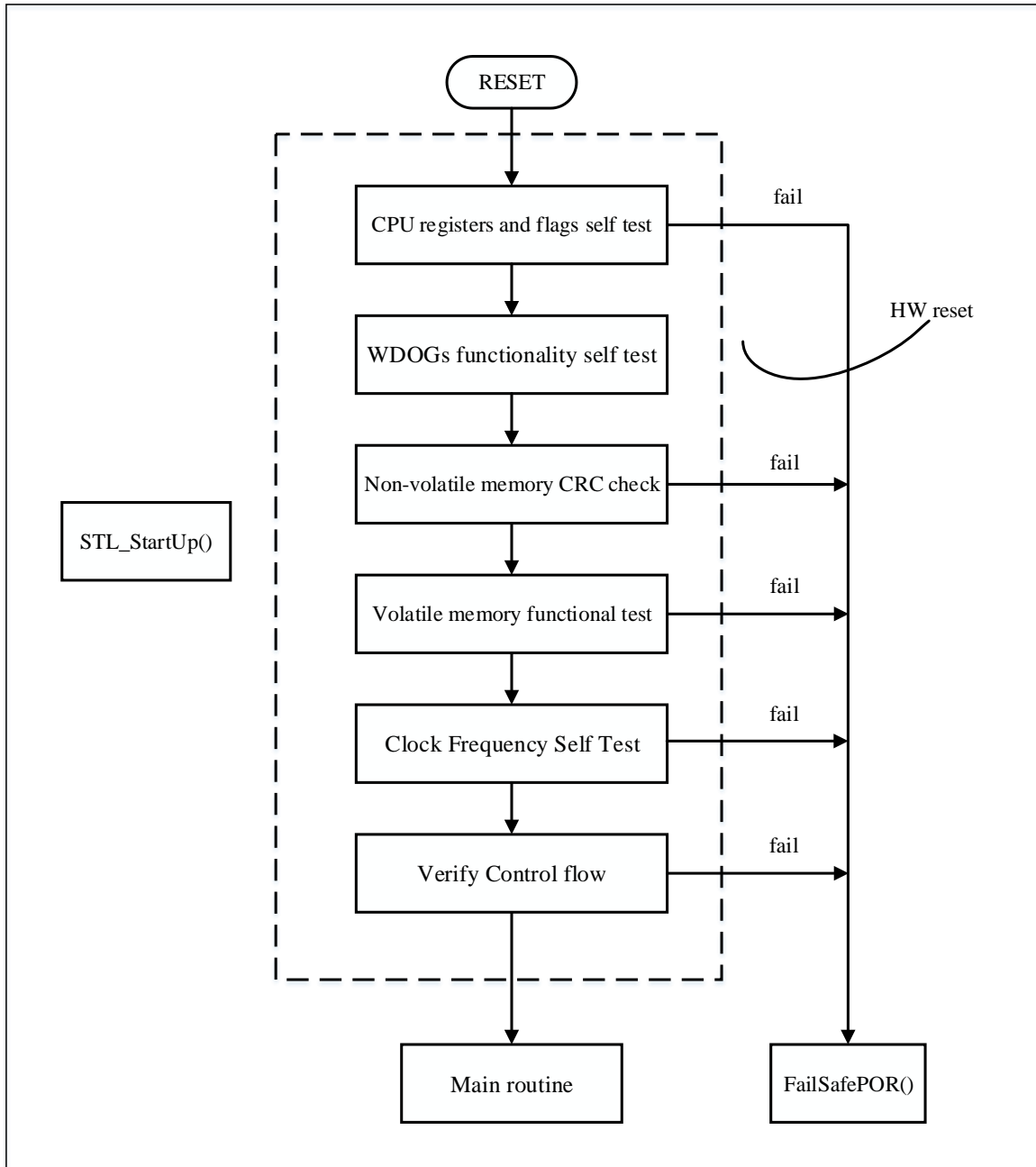
- 局部CPU内核寄存器检测
- 堆栈边界溢出检测
- 系统时钟运行检测
- Flash CRC分段检测
- 看门狗检测
- 局部RAM自检(在中断服务程序中进行)

总体流程框图如下：



3.1 启动时检测流程

在芯片由启动到进入main函数之前，先进行启动检测，修改启动文件来执行该部分代码，检测流程结束后调用__iar_program_start函数跳转回main函数。
下图是执行启动时自检的流程框图：



3.1.1 CPU 启动时检测

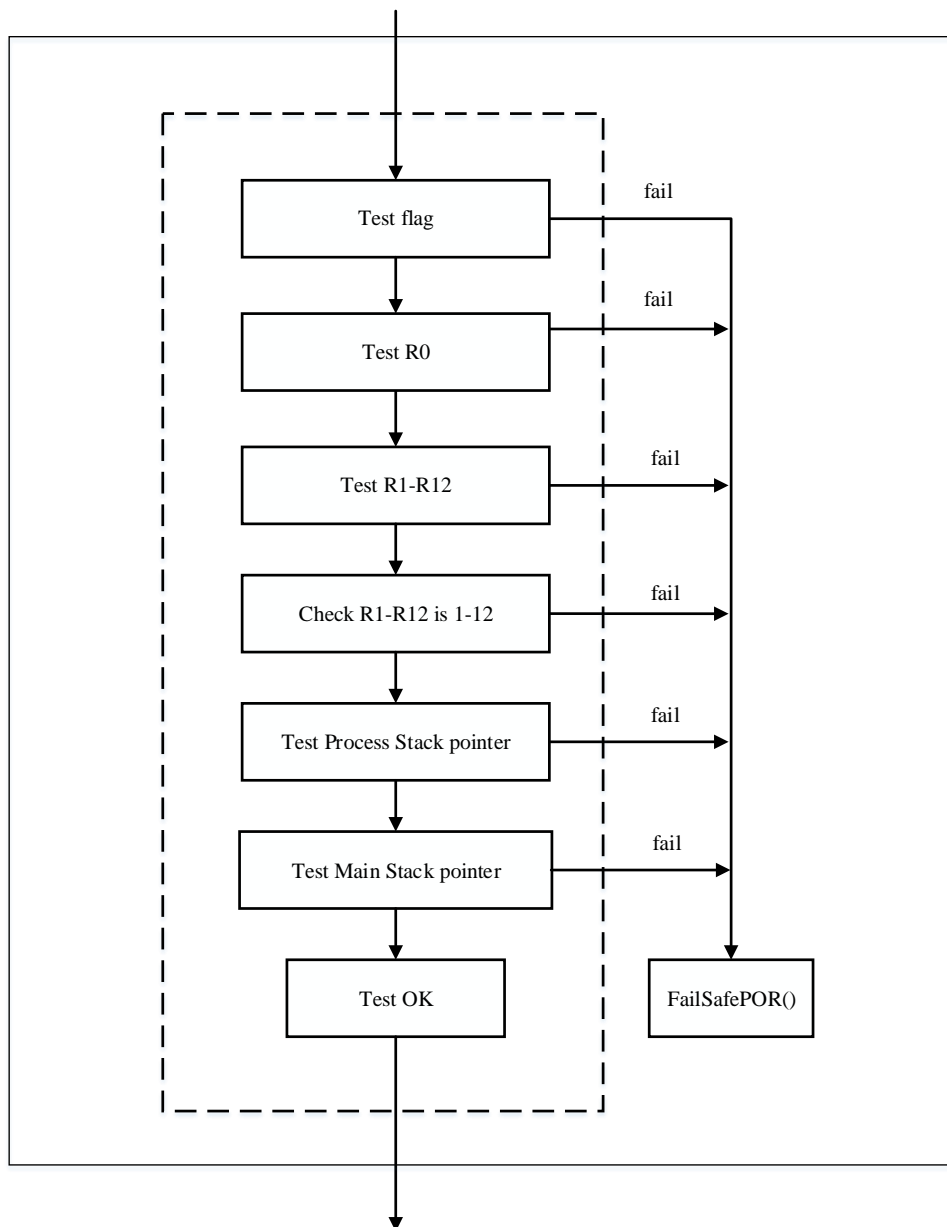
CPU自检主要检查内核标志、寄存器等是否正确。如果发生错误，就会调用故障安全处理函数FailSafePOR ()。

CPU自检在启动时和运行时都会进行，在启动时，R0~R12、PSP、MSP寄存器和Z(zero)、N(negative)、C(carry)、V(overflow)标志位的功能测试都会进行一次自检；运行时，周期性自检，仅检测寄存器R1~R12。

寄存器检测具体实现方式为：分别往寄存器中写入 0xAAAAAAAA 和 0x55555555，再进行比较读取值是否为写入的值。R1 测试完后写 1，R2 测试完后写 2，以此类推。

标志位检测具体实现方式为：分别使标志位置位，若检查标志位错误则进故障函数。

检测流程框图如下：

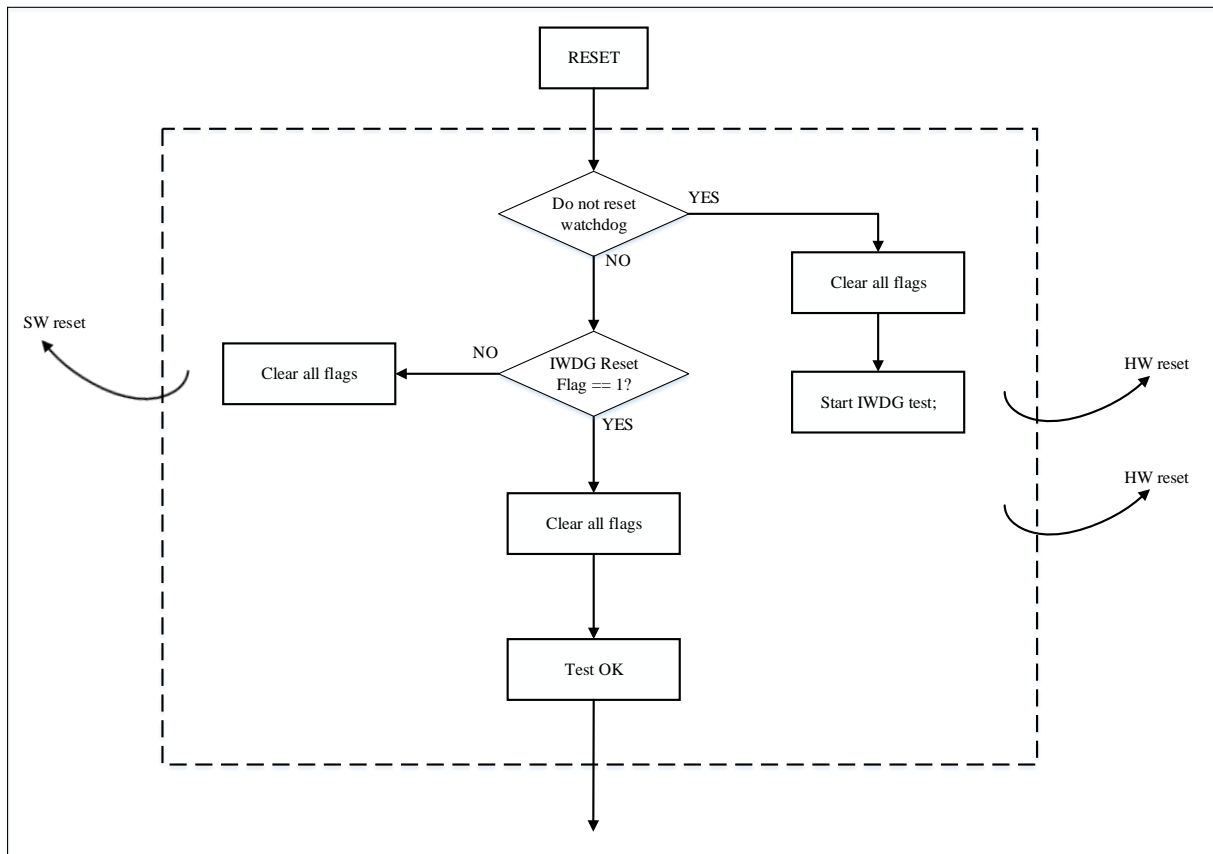


3.1.2 看门狗启动时检测

测试确认独立看门狗能正确复位，以确保在程序运行时跑飞能及时复位防止卡死。

初次复位后清除所有复位状态寄存器标志位，启动 IWDG 测试，使芯片复位，判断是为 IWDG 复位标志位则看门狗测试通过，清除所有标志位。

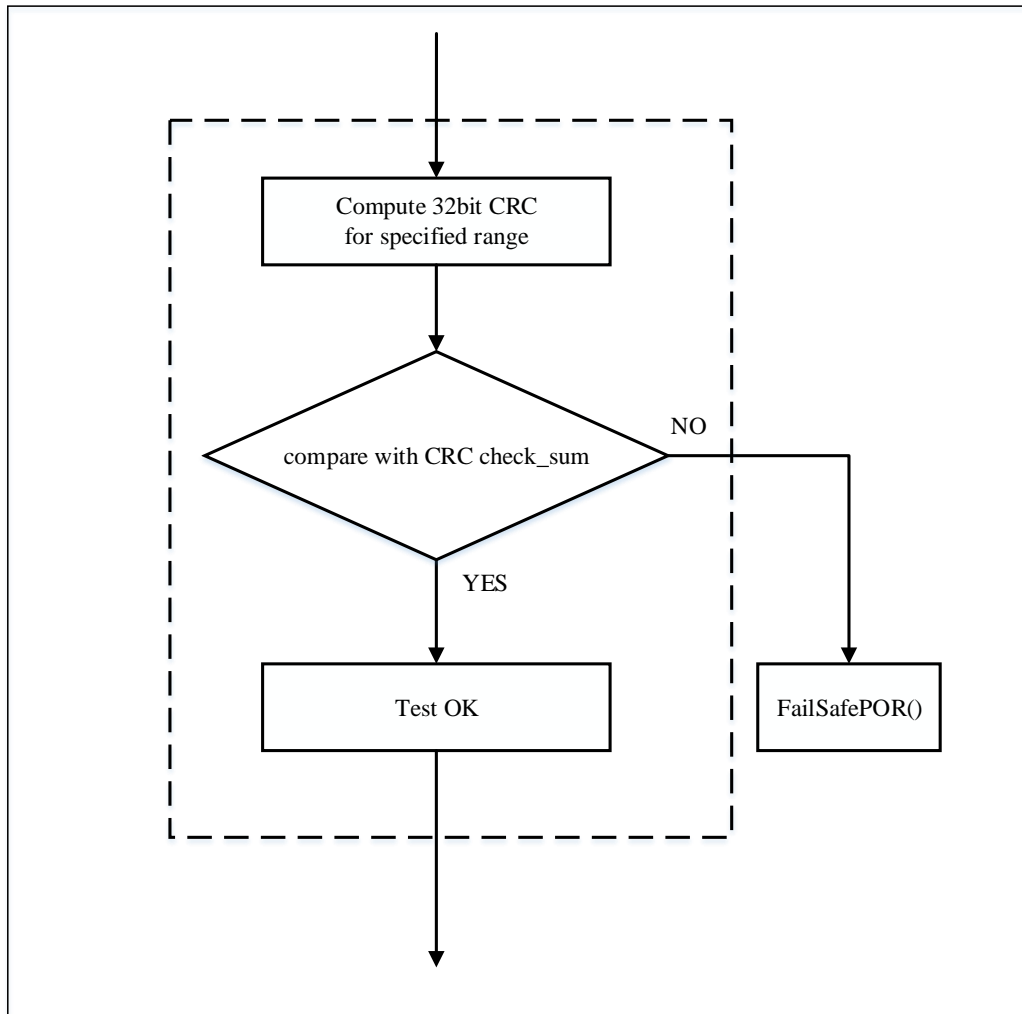
流程框图如下：



3.1.3 FLASH 启动时检测

FLASH自检是程序中将flash数据用CRC算法计算，将结果值跟编译时计算好并且存储在FLASH指定位置的CRC值进行比较，以此确认FLASH完整性。

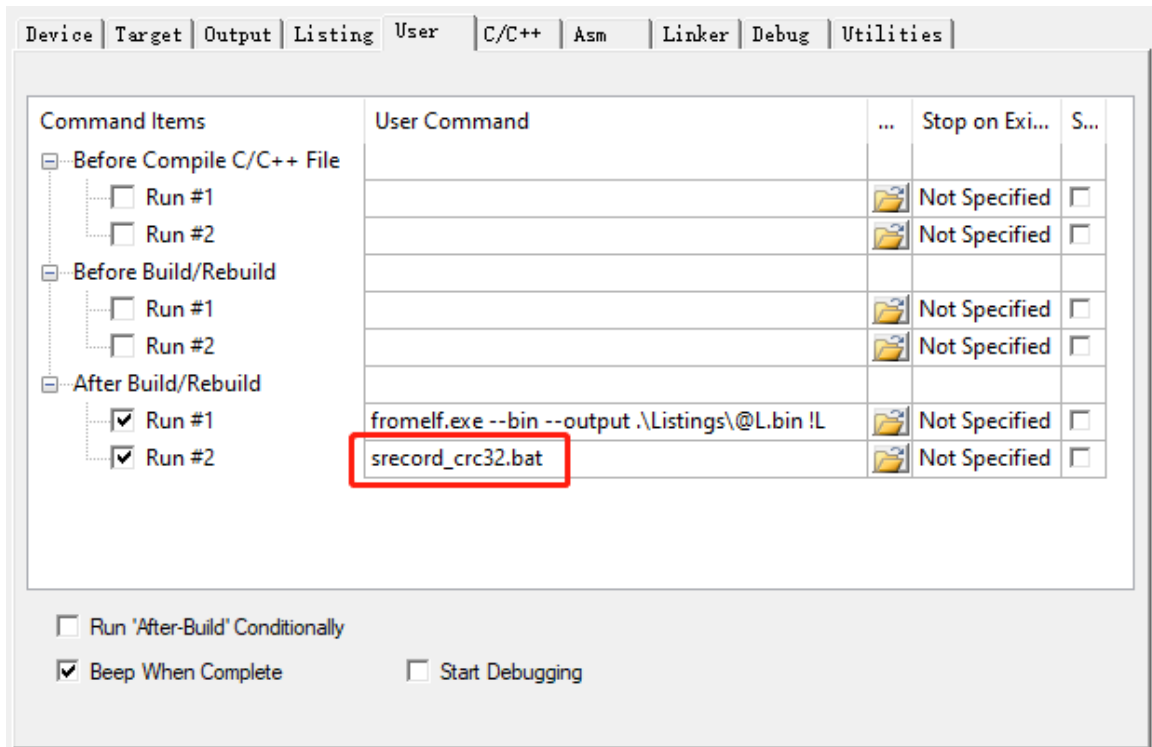
流程框图如下：



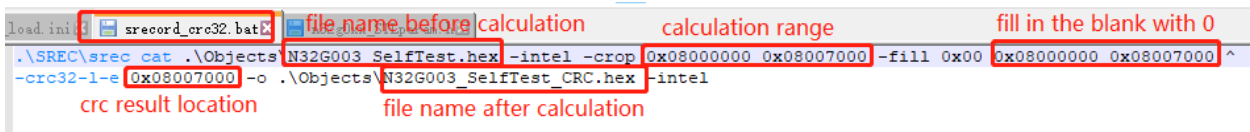
其中CRC计算的FLASH范围根据整个程序的实际情况进行配置：

Keil的配置较为复杂，ARM官方对于ROM Self-Test in MDK-ARM推荐使用第三方软件SRecord进行CRC测试。

根据工程配置，编译完成后会调用脚本文件srecord_crc32.bat，通过srec_cat.exe软件，将Keil编译生成的N32G003_SelfTest.hex文件中的数据进行CRC计算，生成CRC校验结果，添加到指定位置得到新的N32G003_SelfTest_CRC.hex文件：



使用Notepad或其他工具打开.bat文件，根据实际应用修改以下内容：

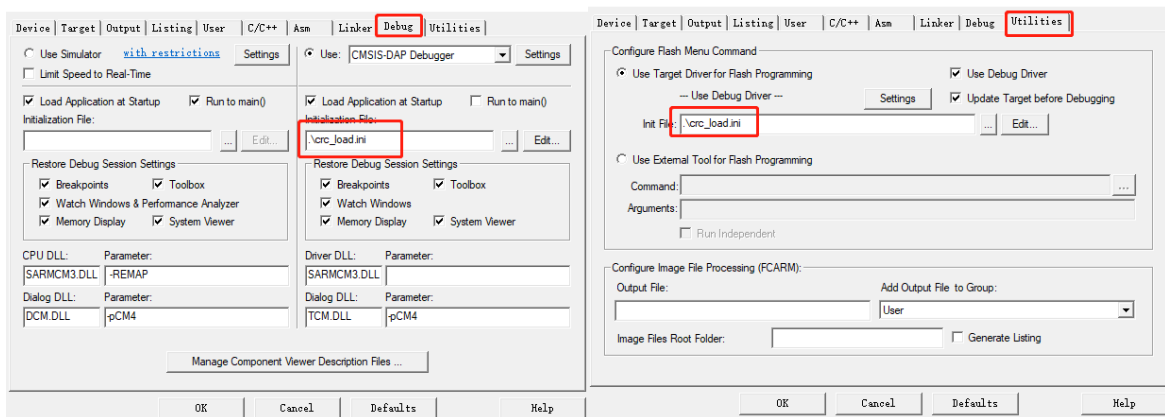


程序中计算CRC的范围根据n32g003_STLparam.h文件配置，可根据需求修改，与以上配置保持一致：

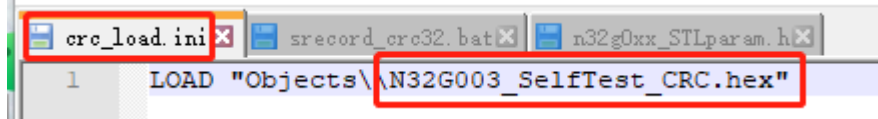
```

/* Constants necessary for Flash CRC calculation (ROM_SIZE in byte) */
/* byte-aligned addresses */
#define ROM_START ((uint32_t *)0x08000000uL)
#define ROM_END ((uint32_t *)0x08007000uL) /* Modify according to needs */
#define ROM_SIZE ((uint32_t)ROM_END - (uint32_t)ROM_START)
    
```

因此不论是在下载还是调试时，都需要使用最终生成的N32G003_SelfTest_CRC.hex文件，所以在Keil配置选项中需添加.ini文件用于下载新的.hex文件，配置如下：



需要注意，.ini文件也要根据实际应用修改内容文件名配置：



```
1 LOAD "Objects\N32G003_SelfTest_CRC.hex"
```

3.1.4 RAM 启动时检测

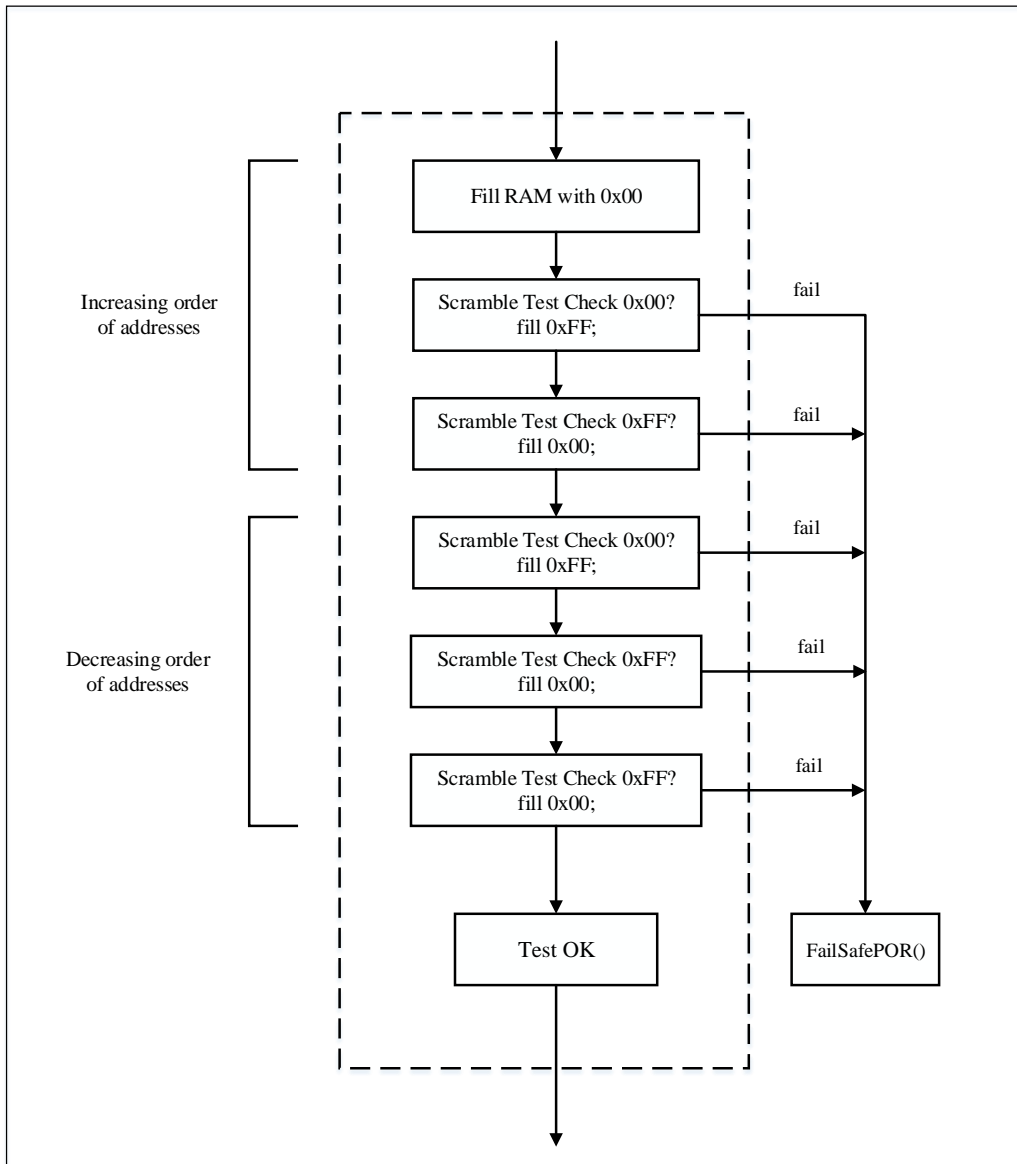
SRAM检测不仅检测数据区域的错误，还检测其内部地址和数据路径的错误。

SRAM自检采用March-C算法，March-C是一种用于嵌入式芯片SRAM测试的算法，是安全认证的一部分。启动时对SRAM所有范围进行检测。

首先将SRAM整片清零，然后按位逐位置1，每置一位，测试该位是不是1，是就继续，不是就报错；全部置完后，再逐位清0，每清一个位，测试该位清0是不是0，如果是就正确，不是就报错。直至完成整个RAM空间的测试

测试时分6个循环，用值0x00和0xFF逐字交替检查和填充整个RAM，前3个循环按照地址递增执行，后3个循环按照地址递减执行。

整个RAM检测算法流程如下图所示：

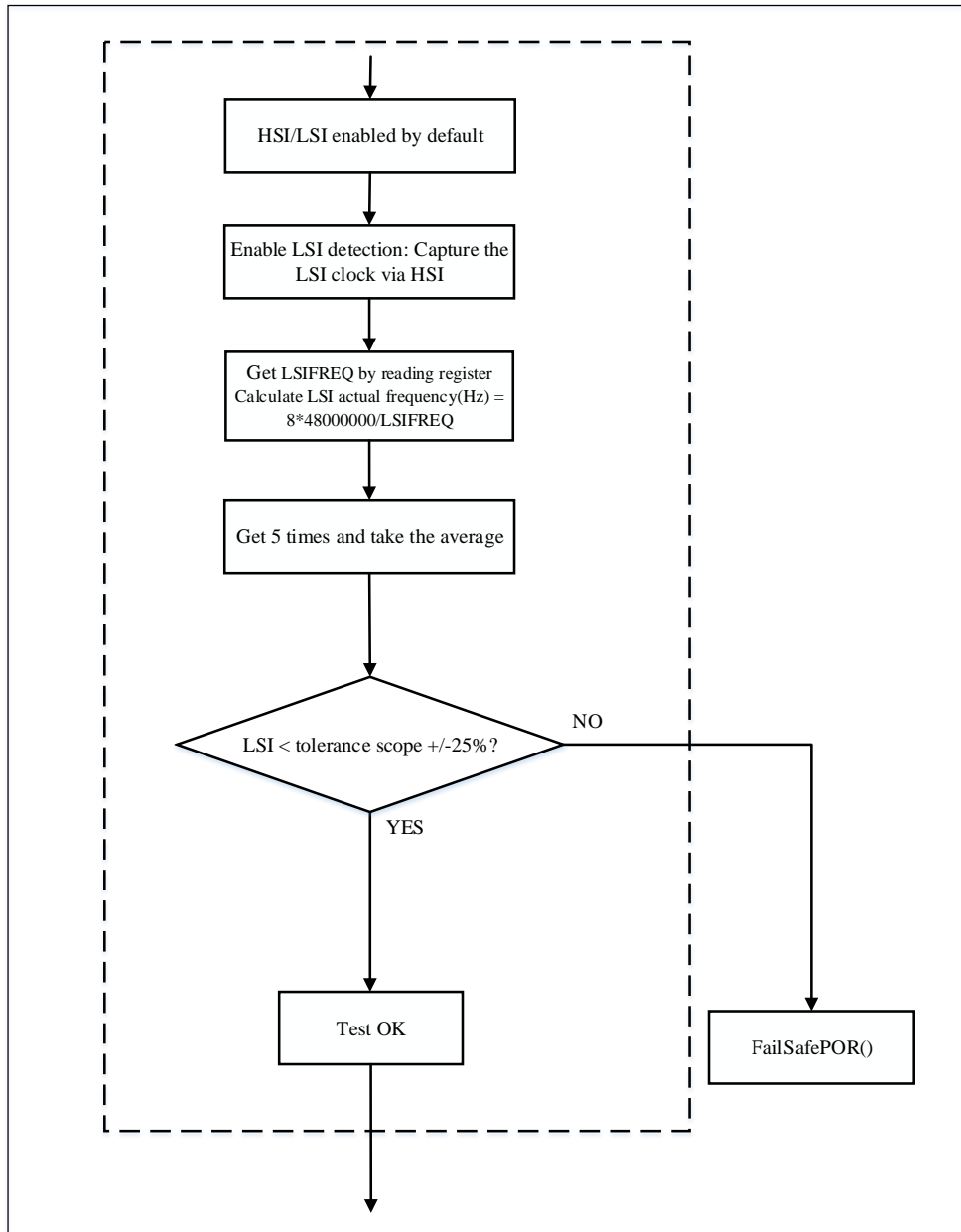


3.1.5 时钟启动时检测

N32G003提供LSI检测功能，通过HSI捕获计算LSI实际频率，测试步骤如下：

1. 使能RCC_LSICTRL.LSIDETEN，启动LSI时钟校准；
2. 查询RCC_LSICTRL.LSIDETFF标志位，为1时校准结束，读取RCC_LSICTRL.LSIFREQ；
3. 根据需要可多次获取RCC_LSICTRL.LSIFREQ后取平均值，然后清除RCC_LSICTRL.LSIDETFF标志位，禁能RCC_LSICTRL.LSIDETEN；
4. 根据公式计算出LSI实际频率 $LSICLK(Hz) = 8 * HSICLK / LSIFREQ$ (HSICLK为48000000)，将该频率值与预期的范围值进行比较：如果超过 $\pm 25\%$ ，则测试失败。

预期范围值可由用户自己根据实际应用情况调整，宏定义为LSI_LimitHigh及LSI_LimitLow。



3.1.6 控制流启动时检测

开机启动自检部分以控制流检测指针程序结束。

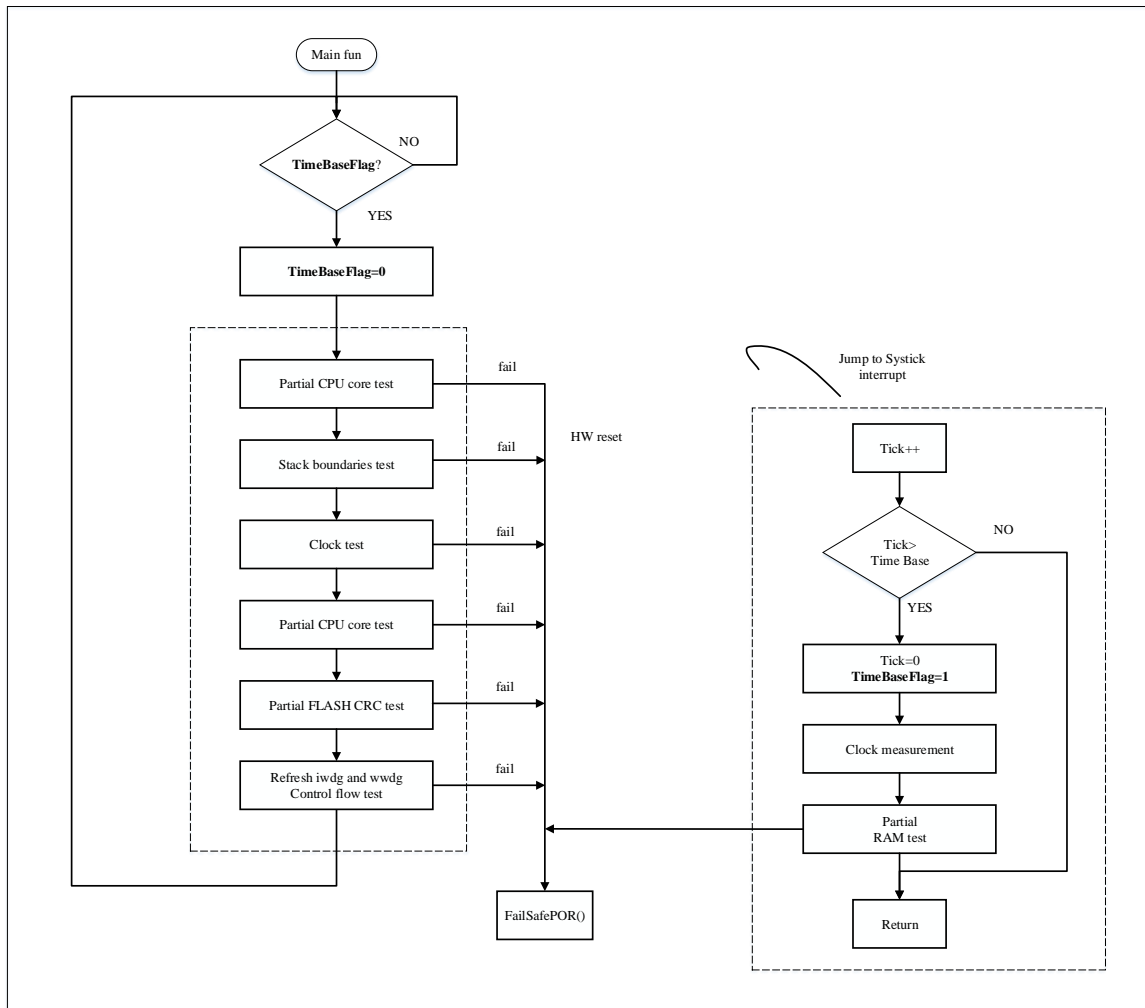
初始化变量 CtrlFlowCnt 为 0， CtrlFlowCntInv 为 0xFFFFFFFF， 每测试一个步骤 CtrlFlowCnt 加一个固定值， CtrlFlowCntInv 减同一固定值， 启动自检结束时判断两个值相加是否仍为 0xFFFFFFFF。

3.2 运行时检测流程

如果启动时的自检成功通过，运行时的周期自检必须在进入主循环之前进行初始化。

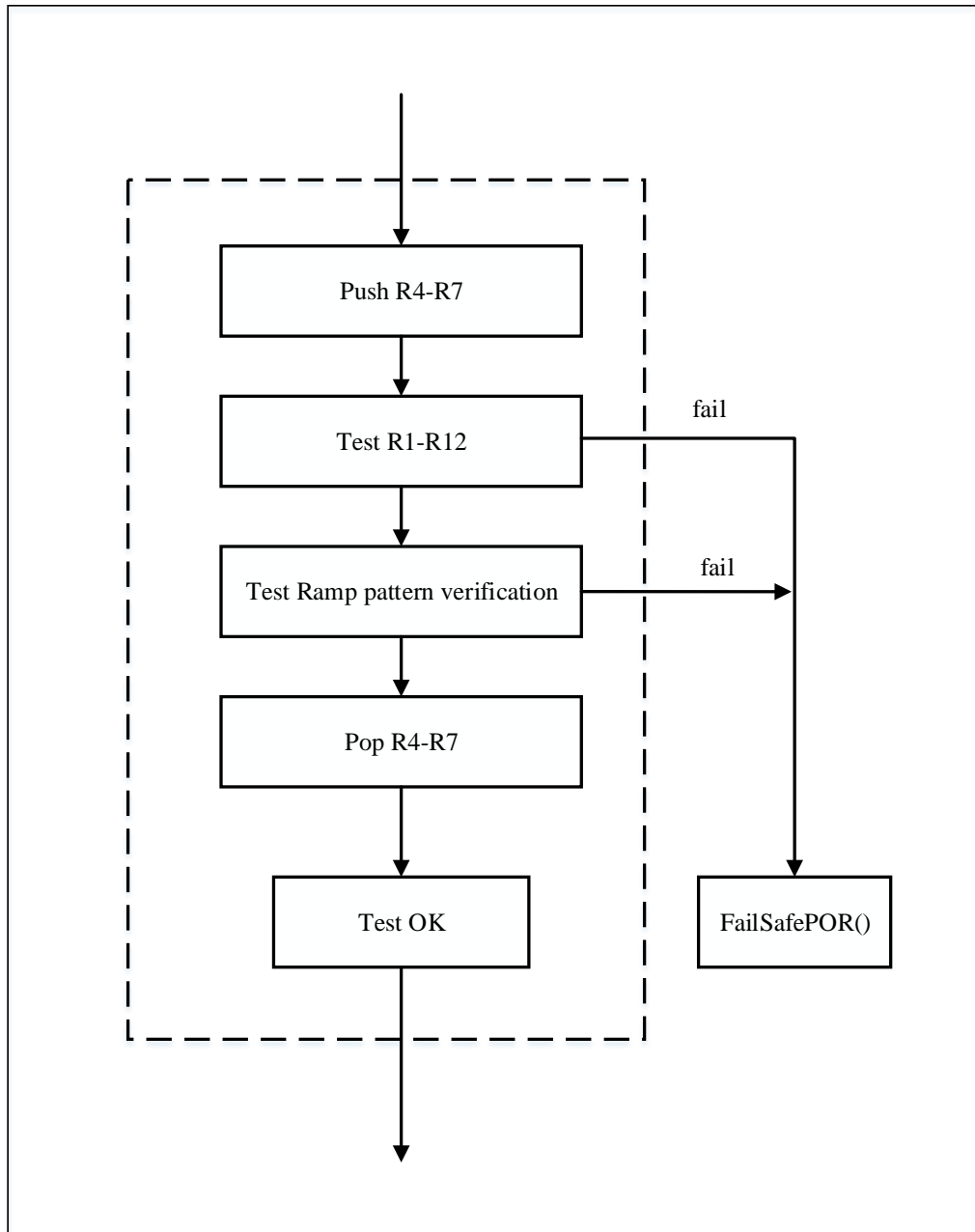
运行时的检测是以 systick 作为时基，进行周期性的检测。

运行时周期检测流程如下：



3.2.1 CPU 运行时检测

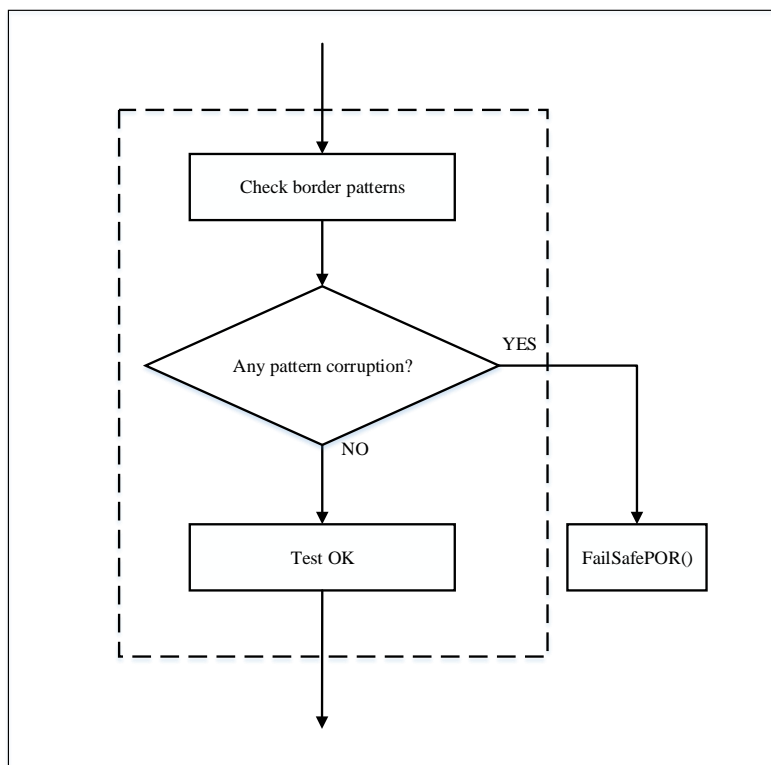
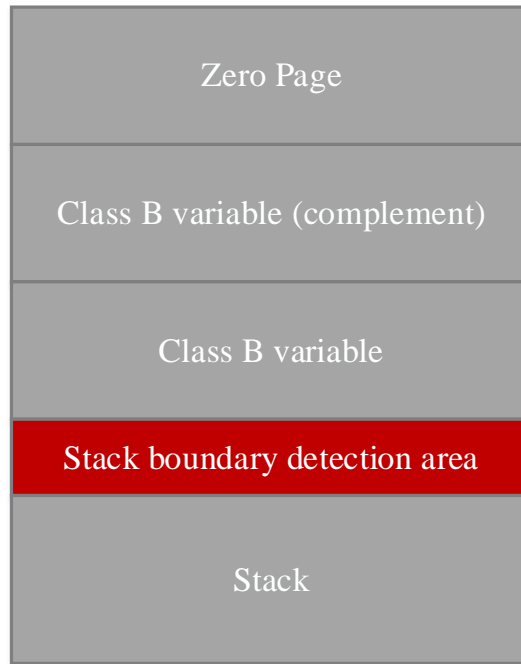
CPU 运行时周期自检跟启动时的自检类似，只是不检测内核标志和堆栈指针。



3.2.2 堆栈边界运行时溢出检测

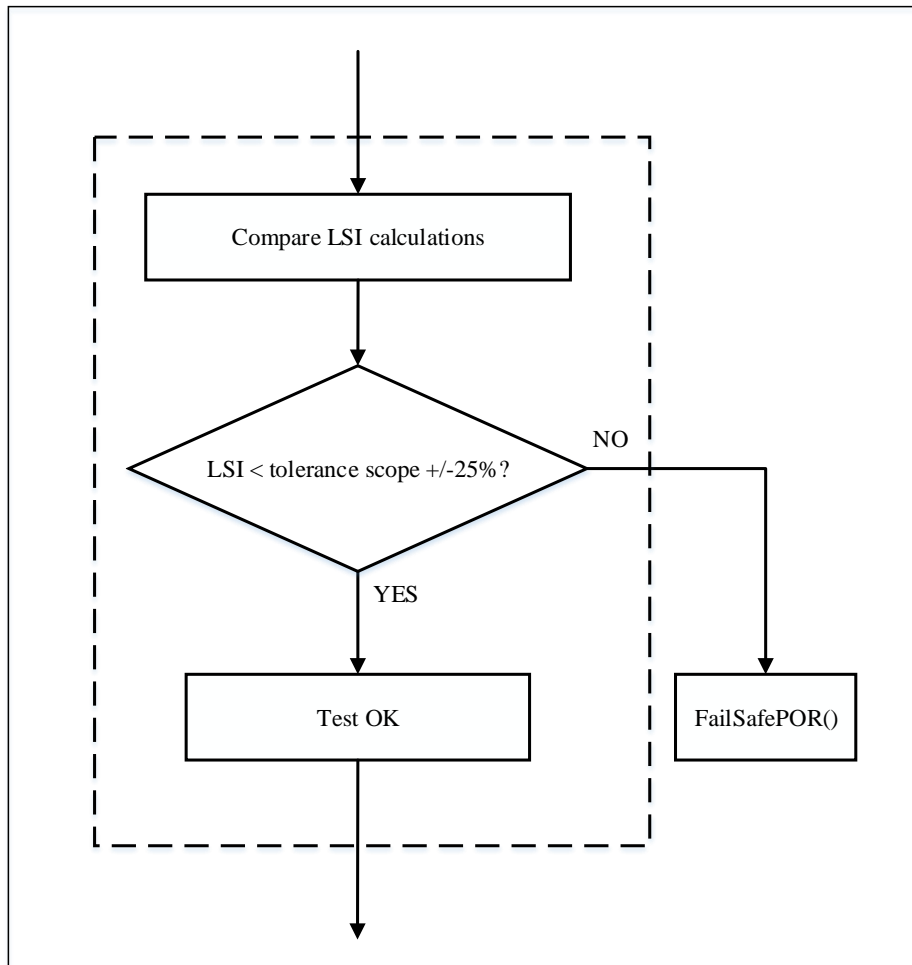
该测试通过判断边界检测区的pattern数组数据完整性来检测堆栈是否溢出。如果原始pattern数据被破坏，则测试失败，调用故障安全程序。

在紧跟堆栈区的低地址位置，定义为堆栈边界检测区。这一区域根据设备可以有不同的配置。用户必须为堆栈定义足够的区域，并保证pattern正确放置。



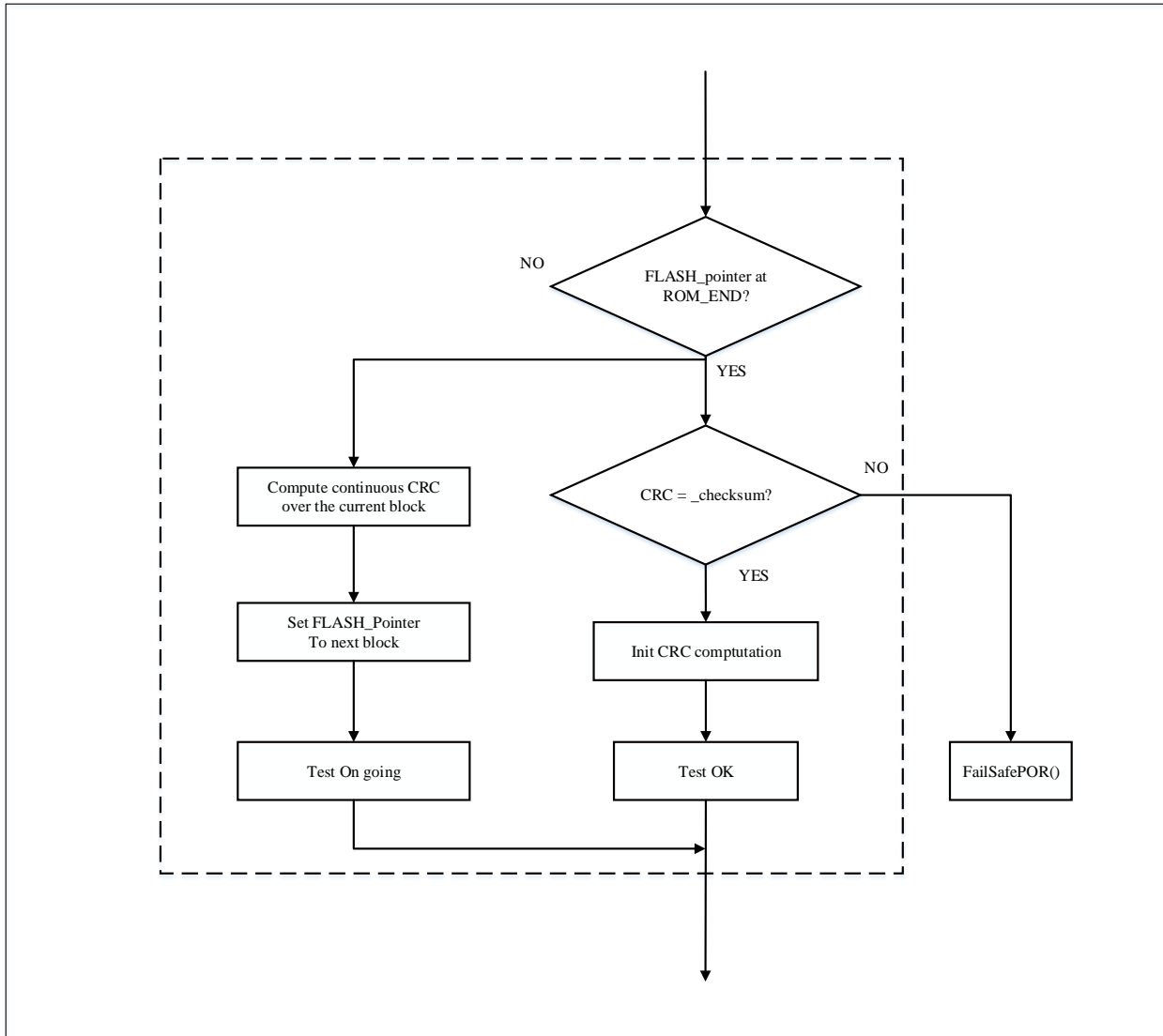
3.2.3 系统时钟运行时检测

运行时系统时钟的检测跟启动时时钟检测类似，直接比较LSI计算值，流程如下：



3.2.4 FLASH 运行时检测

运行时进行Flash CRC的自检，因为检测范围不同耗时不同，可以根据用户应用程序大小配置分段CRC计算，当计算到最后一段范围时，进行CRC值比较，如果不一致则测试失败。



3.2.5 看门狗运行时检测

运行时需要定期喂狗保证系统正常运行，看门狗喂狗的部分放置在 STL_DoRunTimeChecks()最后部分。

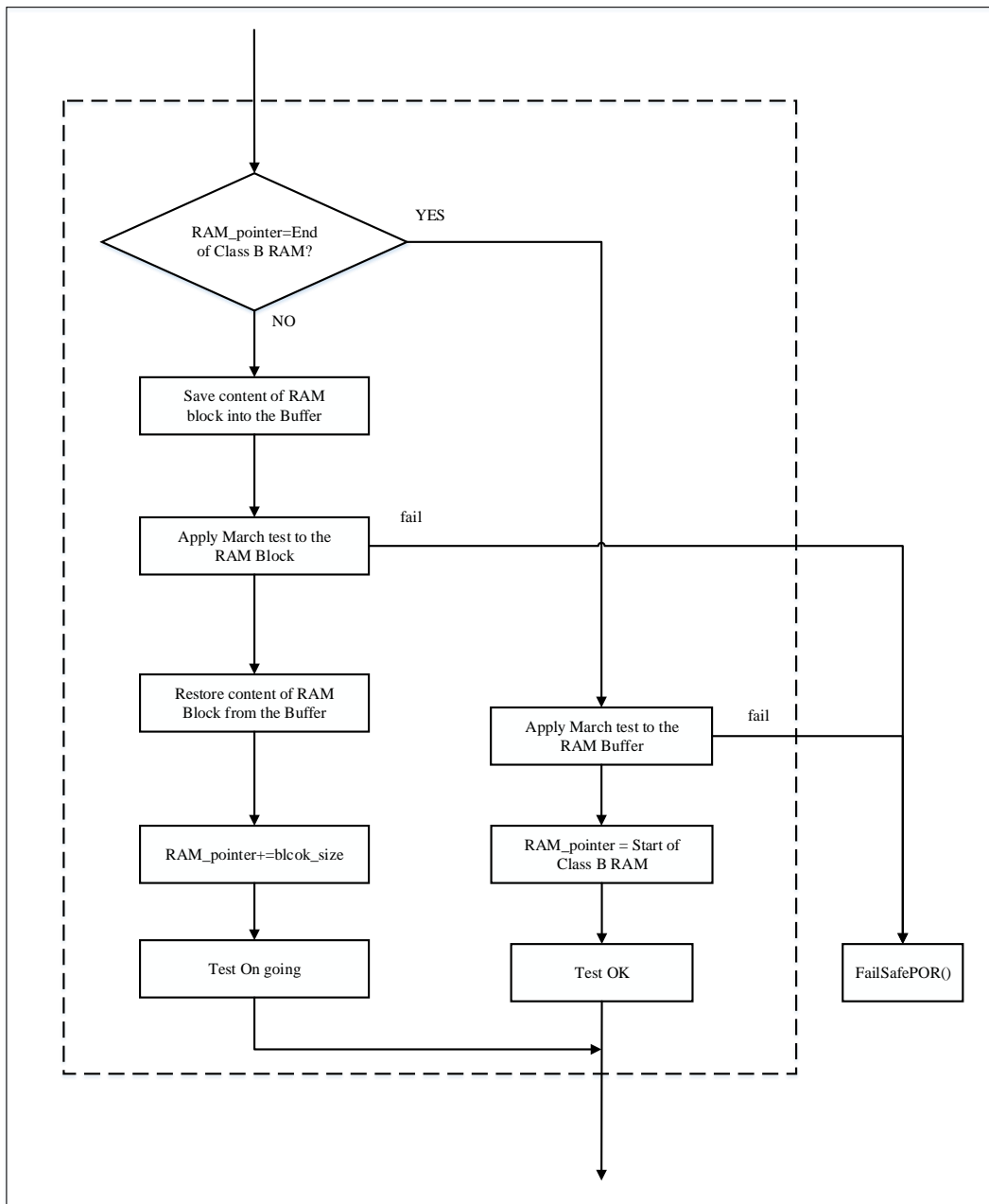
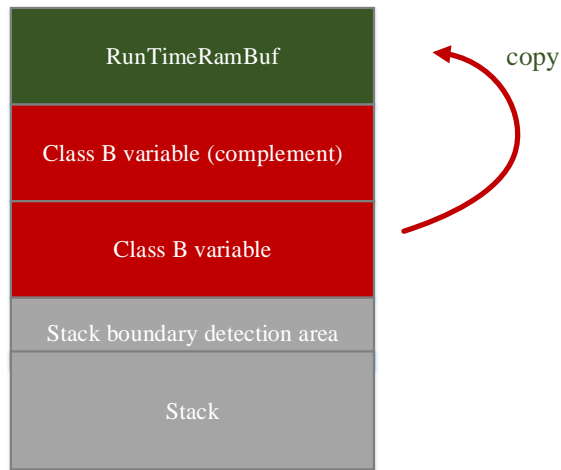
3.2.6 局部 RAM 运行时自检

运行时的RAM自检是在systick中断函数中进行的。测试只覆盖分配给class B变量的那部分内存。

根据class B变量划分的区域，每6字节为一个块，进行March-C测试前将块数据保存在 RunTimeRamBuf中，一块测试完成后再将RunTimeRamBuf放回class B区域原位置。直到 class B区域全部测试完成。

class B区域测试完成后，对RunTimeRamBuf区域进行March-C测试，测试完成则恢复指针至class B开始地址，开始下一次测试。

注意：RAM检测期间会暂时关闭全局中断，结束后放开中断，因此用户需要注意防止应用代码与RAM检测发生冲突，如I2C模块，需要确认数据收发成后再进行RAM检测。



4. 软件库移植要点

- 在执行用户程序之前，先执行STL_StartUp函数（启动自检）；
- 设置IWDG，防止其在程序正常运行时复位；
- 设置启动和运行时的RAM和FLASH检测范围：
 - CRC校验的范围，checksum在Flash中存储的位置
 - ClassB变量的存储地址范围
 - 堆栈边界检测区的位置
- 对检测到的故障进行处理；
- 根据具体的应用，增加用户相关的故障检测内容；
- 根据具体应用定义程序运行时自检的频率；
- 芯片复位后，在执行初始化工作之前，必须先调用STL_StartUp函数进行启动时的自检；
- 在进入main函数主循环前调用STL_InitRunTimeChecks()，主循环中调用STL_DoRunTimeChecks()；
- 用户可以放开Verbose注释进入诊断模式，通过UART1的Tx pin(PA2)脚输出文字信息。

串口配置为115200Bd, no parity, 8-bit data, 1 stop bit;

- 自检库占用内存情况如下：

```
=====
Total RO Size (Code + RO Data)          9396 (  9.18kB)
Total RW Size (RW Data + ZI Data)       1212 (  1.18kB)
Total ROM Size (Code + RO Data + RW Data) 9484 (  9.26kB)
=====
```

5. 历史版本

版本	日期	备注
V1.0	2023-01-03	创建文档
V1.1.0	2026-03-06	1. 更新应用笔记代码位置并添加 2 章节说明 2. 3.2.6 章节添加 RAM 检测注意点

6. 声明

国民技术股份有限公司（下称“国民技术”）对此文档拥有专属产权。依据中华人民共和国的法律、条约以及世界其他法域相适用的管辖，此文档及其中描述的国民技术产品（下称“产品”）为公司所有。

国民技术在此并未授予专利权、著作权、商标权或其他任何知识产权许可。所提到或引用的第三方名称或品牌（如有）仅用作区别之目的。

国民技术保留随时变更、订正、增强、修改和改良此文档的权利，恕不另行通知。请使用者在下单购买前联系国民技术获取此文档的最新版本。

国民技术竭力提供准确可信的资讯，但即便如此，并不推定国民技术对此文档准确性和可靠性承担责任。

使用此文档信息以及生成产品时，使用者应当进行合理的设计、编程并测试其功能性和安全性，国民技术不对任何因使用此文档或本产品而产生的任何直接、间接、意外、特殊、惩罚性或衍生性损害结果承担责任。

国民技术对于产品在系统或设备中的应用效果没有任何故意或保证，如有任何应用在其发生操作不当或故障情况下，有可能致使人员伤亡、人身伤害或严重财产损失，则此类应用被视为“不安全使用”。

不安全使用包括但不限于：外科手术设备、原子能控制仪器、飞机或宇宙飞船仪器、所有类型的安全装置以及其他旨在支持或维持生命的应用。

所有不安全使用的风险应由使用人承担，同时使用人应使国民技术免于因为这类不安全使用而导致被诉、支付费用、发生损害或承担责任时的赔偿。

对于此文档和产品的任何明示、默示之保证，包括但不限于适销性、特定用途适用性和不侵权的保证责任，国民技术可在法律允许范围内进行免责。

未经明确许可，任何人不得以任何理由对此文档的全部或部分进行使用、复制、修改、抄录和传播。