AN0041
应用笔记

AT32 IEC 60730 CLASSB软件库使用指南

前言

这篇应用笔记描述了AT32系列的MCU如何执行IEC 60730中所要求的软件安全性相关的操作。

支持型号列表：

支持型号	AT32 全系列

目录

表目录

图目录

1 概述

IEC60730的附录H(H.2.22)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。

B类软件：软件的设计要防止电子设备的不安全操作。

C类软件：软件的设计为了避免某些特殊的危险。

本文主要介绍针对其中B类软件而开发的CLASSB软件包，安全库相关代码基本是独立于芯片外设IP，软件包中各型号的example都是基于AT-START开发板，其余暂未支持型号用户可自行参考移植。

2 软件库总体架构

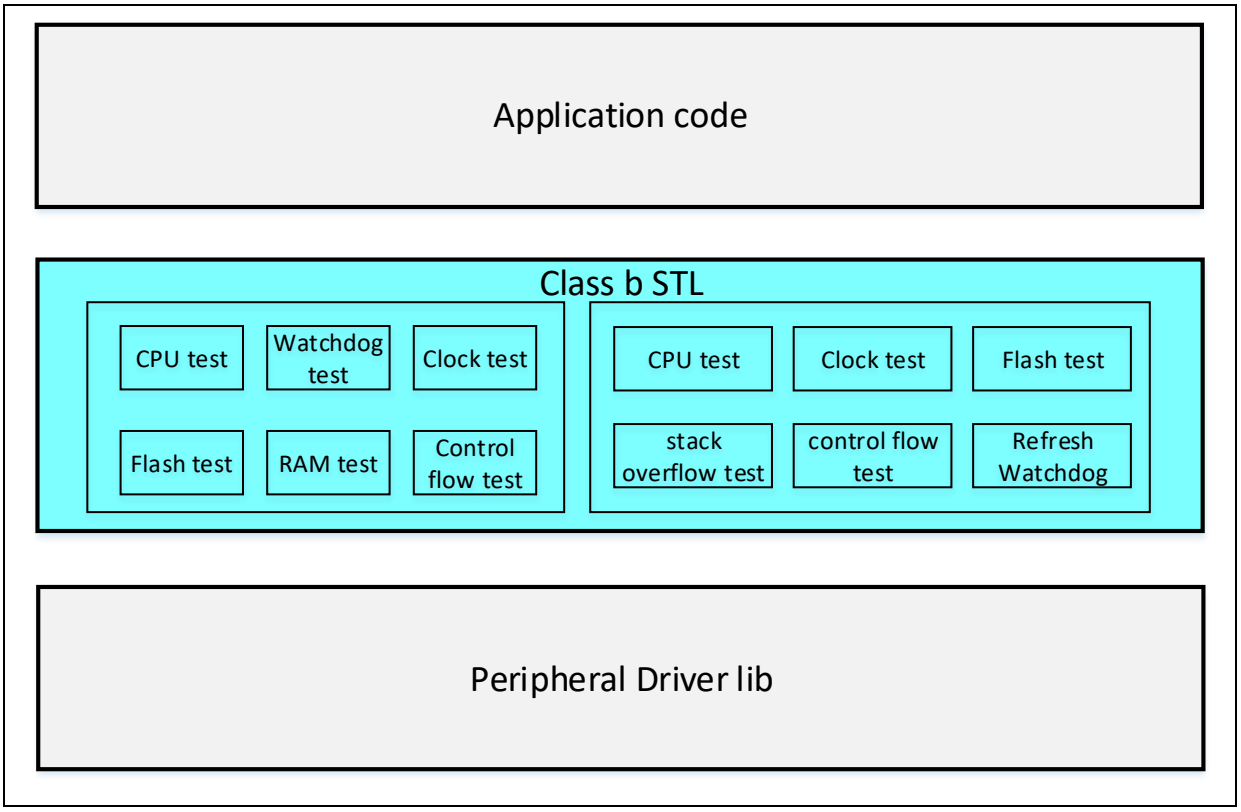
Class B检测诊断库软件包，基于AT32芯片的V2版本固件库的文件夹结构，删除了原固件库中的demo、middlewares、dsp lib等IEC60730不需要的部分，添加集成IEC60730安全检测相关代码。软件包文件夹总体结构框架如下表：

表 1 Class B 软件包文件夹结构

主目录	驱动目录	备注
libraries	cmsis	内核相关
	drivers	芯片外设相关驱动
middlewares	classb_lib	通用部分的STL程序
project	at32xxx_board	评估板配置集成代码
utilities	classb_demo	跟产品、开发环境、工程相关的STL程序

STL软件由芯片外设驱动libraries、class b STL和user application组成，其中class b STL分为两个主要部分：启动时检测和运行时周期检测，架构整体框图如下：

表 2 软件架构整体框图



3 软件库设计

本章节介绍STL例程通用的基本原理，并对工程结构以及配置和调试相关信息进行说明。各个开发环境（IAR、Keil、AT32IDE）会有一些差别，也进行了说明。

3.1 故障安全处理

当自检程序检测到故障时，故障安全处理函数selftest_fail_handle()被调用。at32_selftest_startup.c文件中定义了该函数。该程序的目的是提供自定义故障处理接口并允许用户立即做出反应。

例程内部对该函数没有特定处理，只输出调试打印信息，然后执行系统复位。该程序的内容由用户根据实际应用自行开发，在该函数执行应用必须的操作，以保证应用处于安全的状态。

该函数从程序的不同故障位置进行调用，为判断故障问题的严重程度，用户可以重新定义该函数，添加一个特定的输入参数（预先定义好的简单常量）用于区分不同的故障。

3.2 程序 RAM 分布

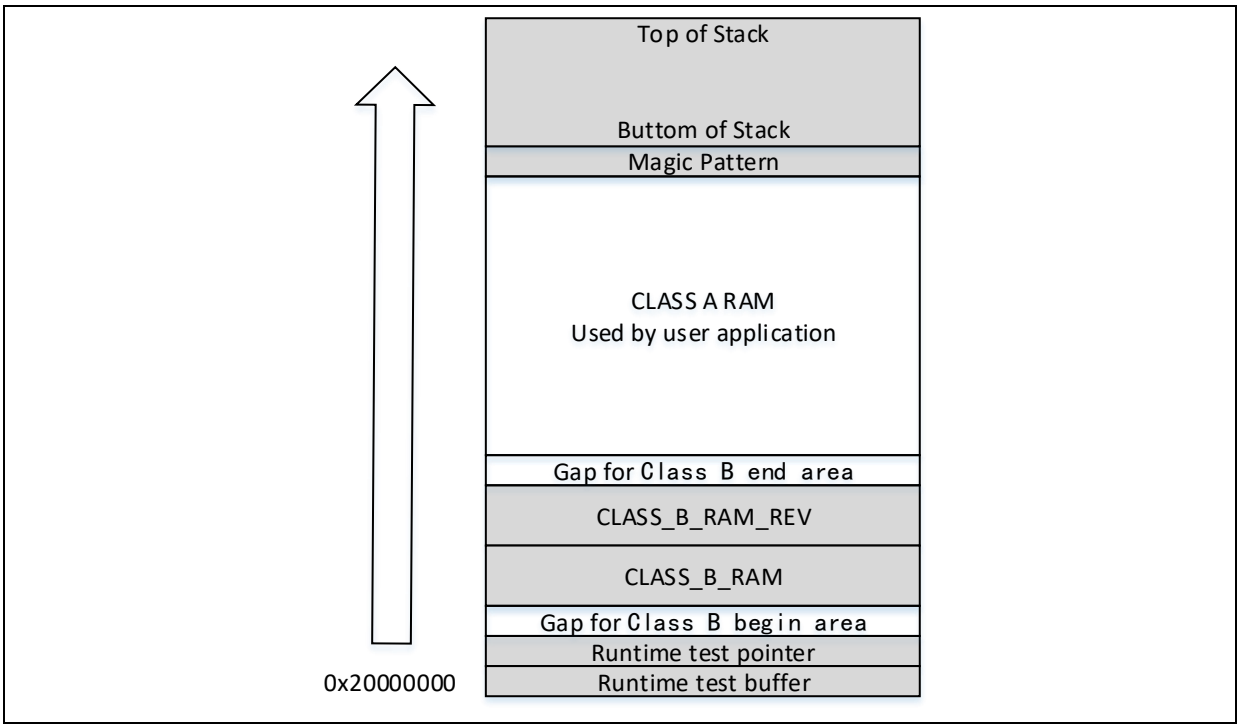
为便于描述，将IEC60730必须用到的相关变量命名为Class B变量，用户应用相关的变量命名为Class A变量。每个Class B变量都使用一对互补值储存在两个单独的RAM区。正常值和补码值始终放在不相邻的内存位置。指定具体内存地址的变量定义根据开发编译环境的不同而写法不同，具体的实现代码通过宏定义区分，定义在at32_selftest_startup.c文件。

- IAR通过修改xxx.icf链接文件指定固定地址
- Keil通过修改xxx.sct分散加载文件指定固定地址
- AT32 IDE通过修改xxx.ld链接文件指定固定地址

每次使用Class B变量数值前，用户必须确保比较每一个数值对（正常值和补码值）的完整性。如果发现任何数值对的完整性被损坏，应调用故障安全处理函数。如果变量值变化，需要同步更新存储位置的补码值以保持正确的数值对。

下图是STL程序中对于RAM区域使用情况的分布图：

图 1 RAM 分布图



3.3 工具特定集成设置

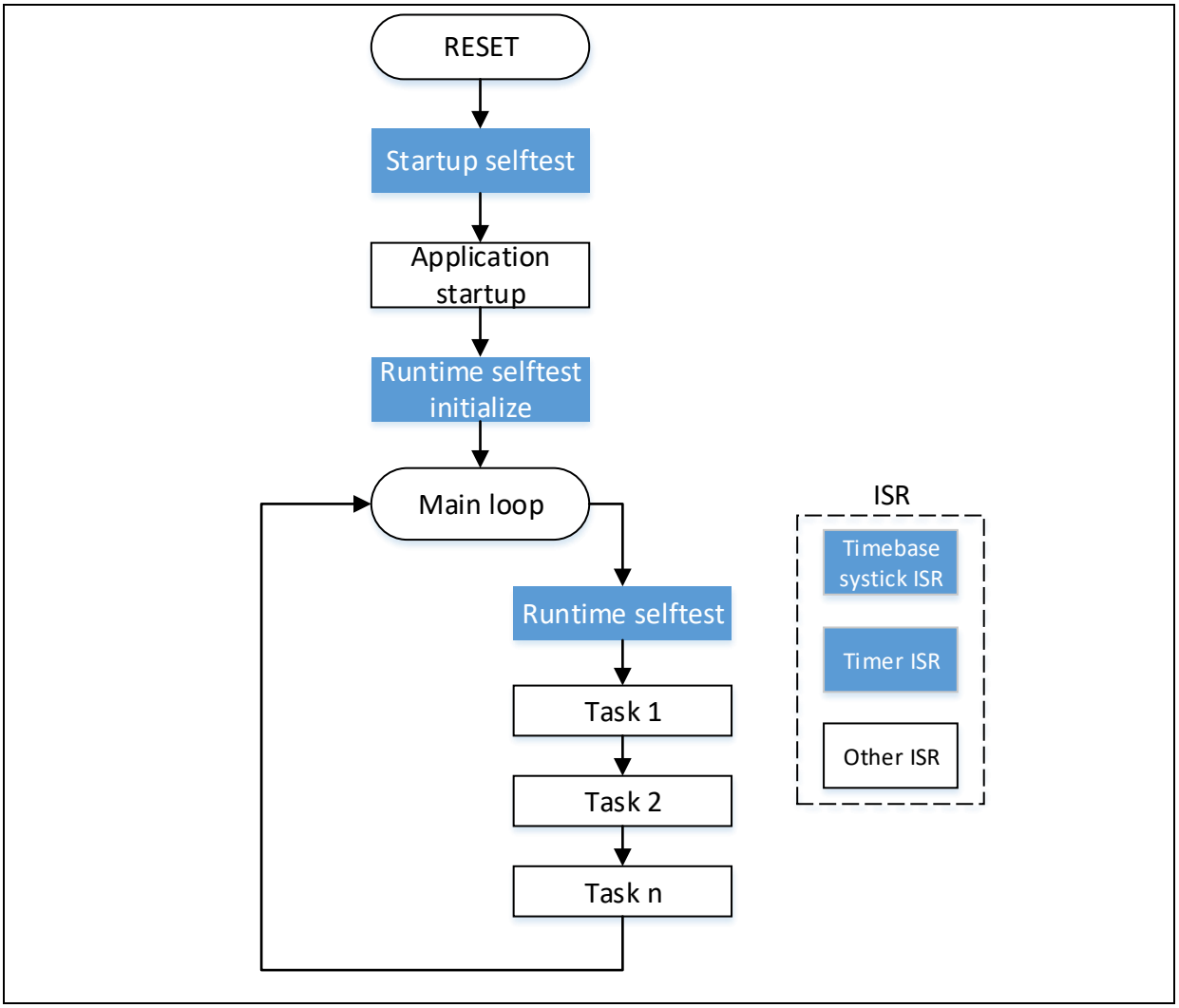
目前STL支持IAR、Keil和AT32 IDE三种IDE，对于不同的IDE，以下事项需注意：

- 修改链接脚本文件*.icf（IAR）、*.sct（Keil）、*.ld（AT32 IDE），定义RAM区域的使用分布情况和CRC checksum位置。
- 启动时的自检代码函数selftest_startup_check()，需要进入主程序前，进行处理调用。IAR和AT32 IDE编译器都是通过修改固件库内对应的原始启动文件startup_at32xxx.s，Keil编译器通过修改$Sub$$main()程序。
- 执行完启动时的自检程序后，需要跳转进入标准C的main主函数，因为不同编译器方式不同，所以用宏goto_compiler_startup()进行定义。
- 在Keil中调用goto_compiler_startup()时，通过ENTER_MAIN_FLAG宏定义的CRC外设的CDT寄存器值，来判断是进入启动自检程序还是真正的mian主程序。

4 程序执行流程

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，总体流程框图如下，图中蓝色框图是相对于原应用程序，执行Class B需添加的部分。

图 2 程序流程总框图



启动时的自检必须在应用启动之前执行，调用函数selftest_startup_check()，在进入主循环前先调用函数selftest_runtime_init()做周期自检初始化配置，然后函数selftest_runtime_check()进行周期自检。例程中周期性自检时基是采用1ms的systick中断，根据变量time_base_flag判断是否进行检测，检测时间间隔由宏定义SYSTICK_10MS_TB决定，用户可以根据自己应用进行调整，例程中是每10ms执行一次周期自检。

注意：运行时如果自检程序耗时太长，会影响正常应用程序的进行，可以将自检流程进行拆分，用systick中断进行周期性的自检。

理论上，自检模块集成添加到应用程序中时，用户需要提供以下步骤：

- 在用户程序启动之前，执行启动前的初始检测
- 在用户程序执行过程中，进行周期性的检测
- 程序运行时，设置看门狗并及时喂狗防止复位
- 对RAM和Flash在初始阶段和运行阶段的测试，设置合适的测试区域

- 对于自检测试发生的错误，以及其他hardfault等错误，需要进行合理的安全处理

注意：当在启动时的测试期间，如果程序启用了调试信息，因为一些外设接口驱动程序可能会用到一些变量，在进行完内存测试后，内存中的内容可能会丢失，所以需要用户保证恢复这些变量。

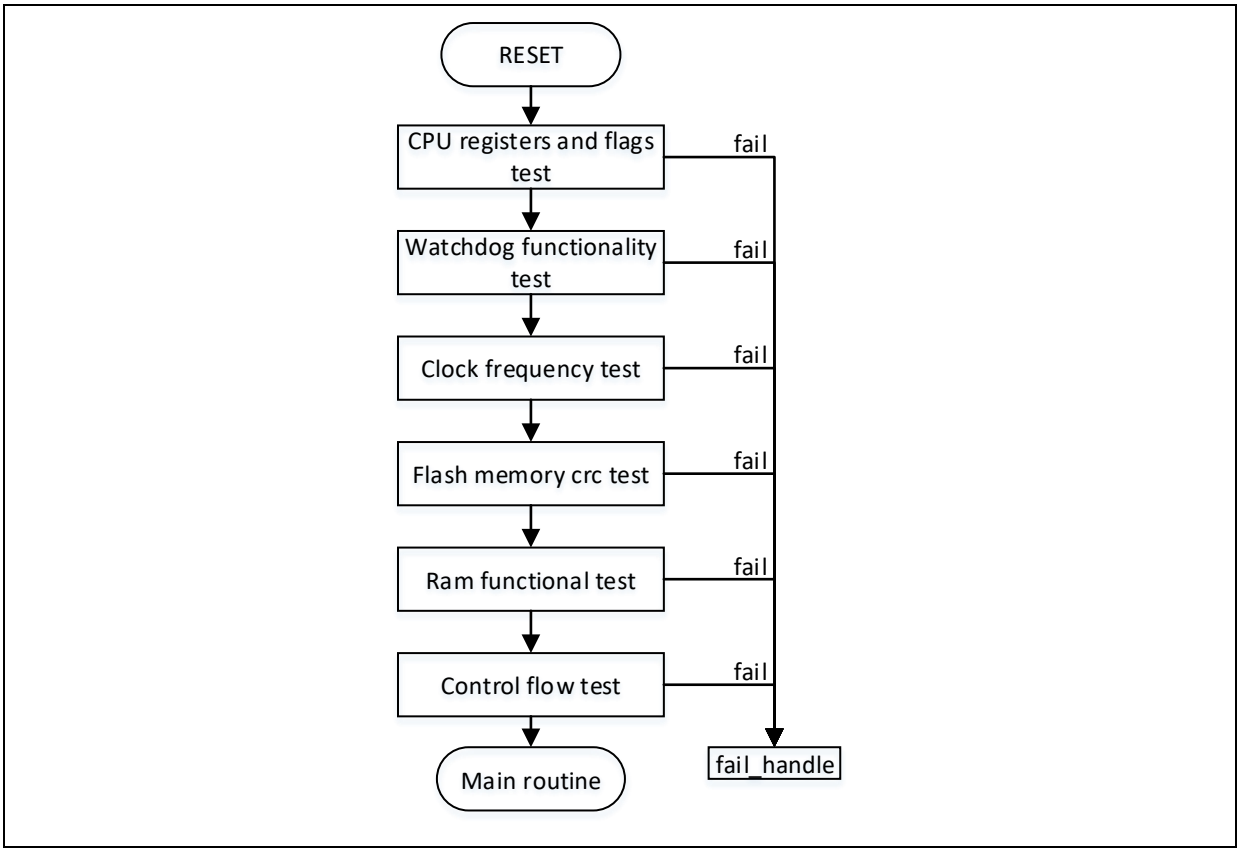
4.1 启动时检测流程

启动时检测包括：

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

下图是执行启动时自检的流程框图。

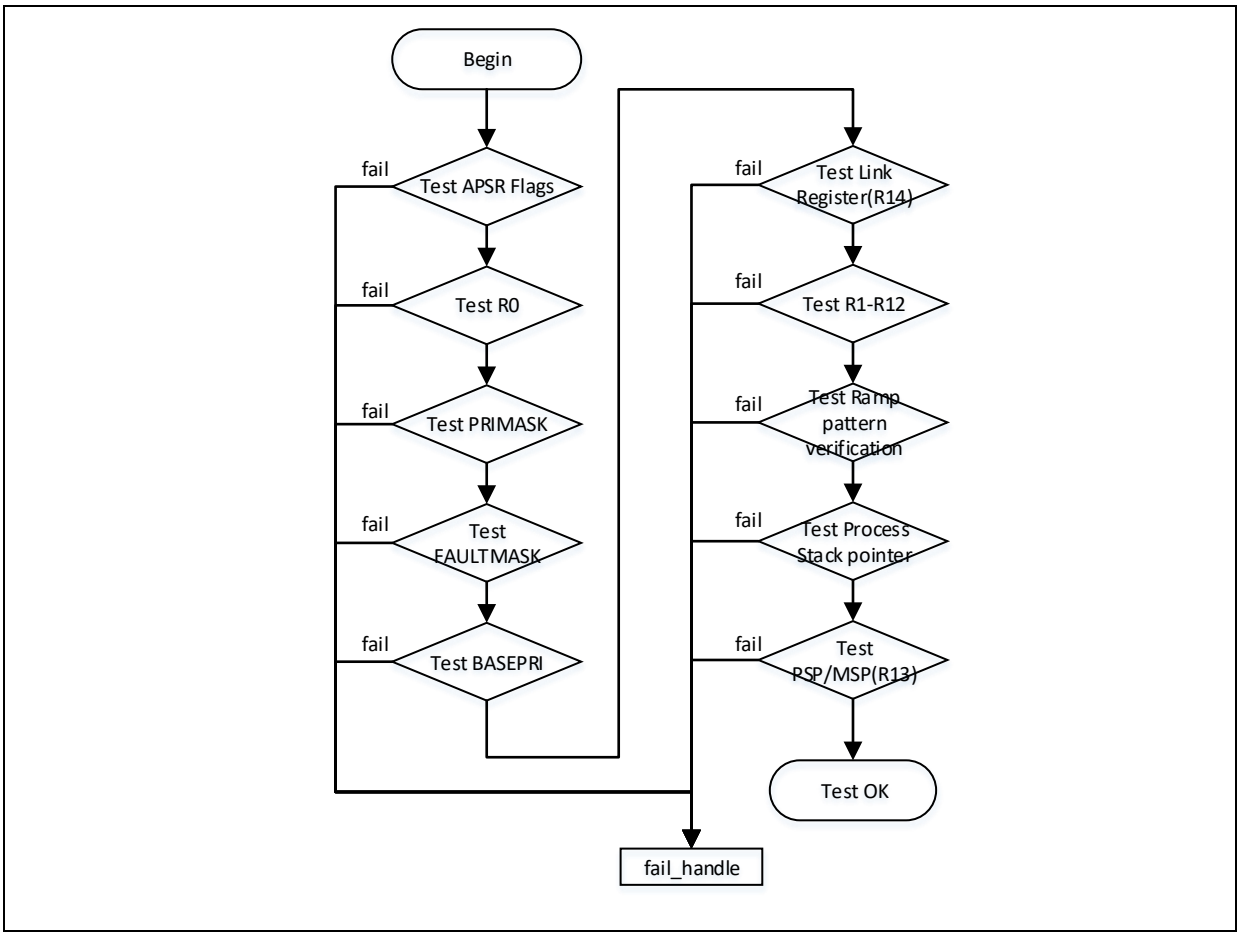
图 3 启动时自检流程结构



4.1.1 CPU 启动时检测

进行CPU寄存器的相关检测，主要检查内核标志、寄存器和堆栈指针等是否正确。如果发生错误，就会调用故障安全处理函数Selftest_fail_handle()。该部分检测源代码是用汇编所写，在KEIL、IAR以及AT32 IDE等不同的编译环境下有差异，另外不同CPU内核调用的指令可能也有差异，具体查看代码。Cortex-M4内核的MCU系列流程框图如下

图 4 cpu 启动时检测流程



4.1.2 看门狗启动时检测

验证看门狗复位功能是否正常，保证后续运行中如果因为程序计数器出现非预期异常情况导致程序跑飞时可以通过看门狗复位恢复。例程中为方便用户参考，宏定义开启了wwdt和wdt，实际应用中，根据需求选择开启某一个或两个都开启。看门狗测试通过判断复位状态寄存器的复位标识来判断是否测试成功，测试流程如下：

A. 判断wdt reset flag和wwdt reset flag是否都没有置起，如果是则执行步骤B开始wdt测试，如果不是则跳转到步骤C

B. 配置wdt并使能，然后不喂狗，等待wdt计数器超时而复位回到步骤A

C. 判断wdt reset flag是否置起，并且wwdt reset flag没有置起，如果是则表明wdt测试通过，执行步骤D开始wwdt测试，如果不是则跳转到步骤E

D. 配置wwdt并使能，然后不喂狗，等待wwdt计数器超时而复位回到步骤A

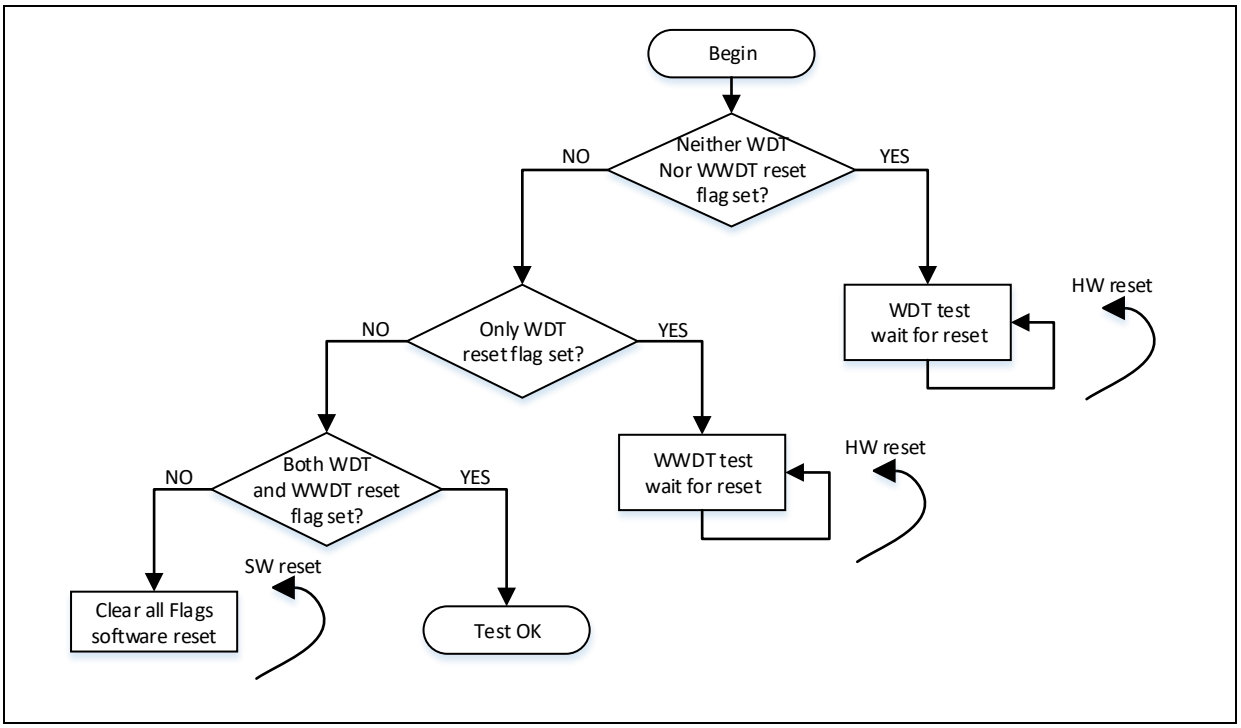
E. 判断wdt reset flag和wwdt reset flag是否都置起，如果是，则表明wdt和wwdt都测试通过，完成整个测试，如果不是则表明测试过程中有未知异常情况，清除所有reset flag，然后执行软件复位，回到步骤A重新开始测试。

注意：为减少启动阶段的测试时长，启动时的看门狗超时时间配置为最小值，运行时的超时时间需要根据实际应用喂狗间隔时间配置。

注意：测试过程中正常情况会有多次系统复位发生，测试完成后需清除所有的复位标志位。

流程框图如下

图 5 看门狗启动时检测流程

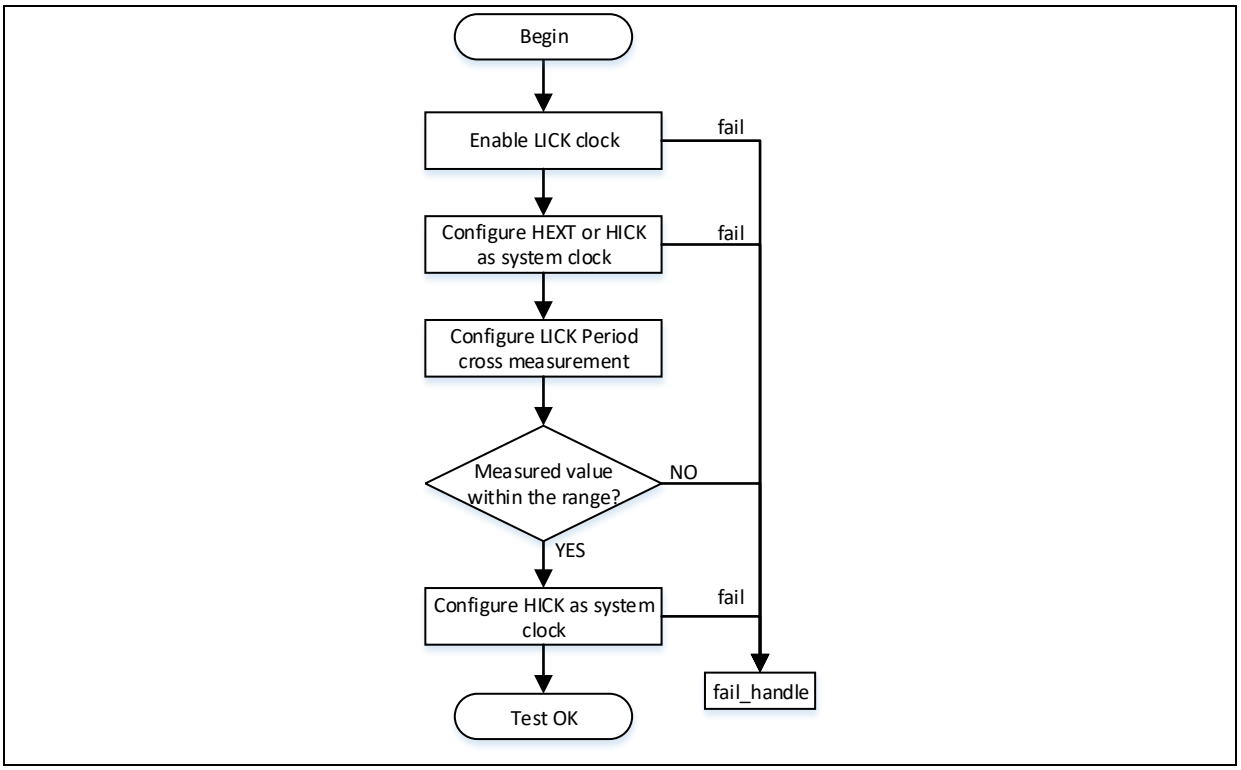


4.1.3 时钟启动时检测

启动阶段的时钟检测通过内部低速时钟源（LICK）和系统时钟的交叉测量结果来验证。系统时钟频率作为基准频率，采用外部高速时钟源（HEXT）或者内部高速时钟源（HICK）。

启动阶段的时钟检测尽量使测试时间短，例程中MCU通常是通过专用的定时器TMR（时钟源为系统时钟）的某个通道输入捕获LICK边沿频率，定时器两次捕获时数据寄存器计数值之间的差值进行计算得到LICK实际测量值，与芯片数据手册上列出的LICK规格的典型值进行比较。如果LICK实际测量值在规格范围的最大和最小范围内，则测试成功；如果超出了规格范围值，则测试失败。

图 6 时钟启动时检测流程

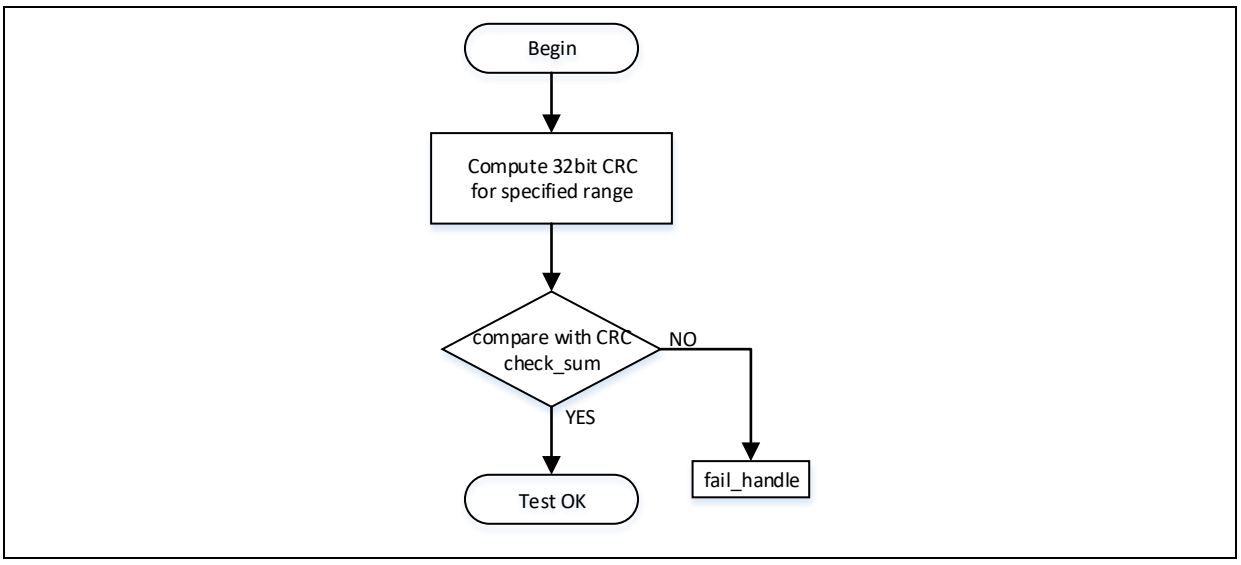


4.1.4 Flash 启动时检测

Flash自检是存储器检测的一部分，程序中测试范围内的Flash数据用CRC外设的算法计算，将结果值跟编译时已存储在Flash指定位置的预先计算好的CRC值进行比较。

Flash的CRC值需要在调试或者烧录阶段跟正常应用代码一起下载到芯片Flash，所以需要在集成开发环境(IDE)编译生成的HEX或者BIN档中进行添加，下面分别介绍Keil、IAR、AT32 IDE，如何增加CRC到已编译的HEX或者BIN文件中，因为Flash CRC运行时周期检测是每次128字节，所以该处设置的CRC地址范围需128字节对齐。

图 7 Flash 启动时 CRC 检测流程



注意：CRC计算的Flash范围需根据整个程序的实际情况进行配置，配置方法在各个IDE上有所不同。

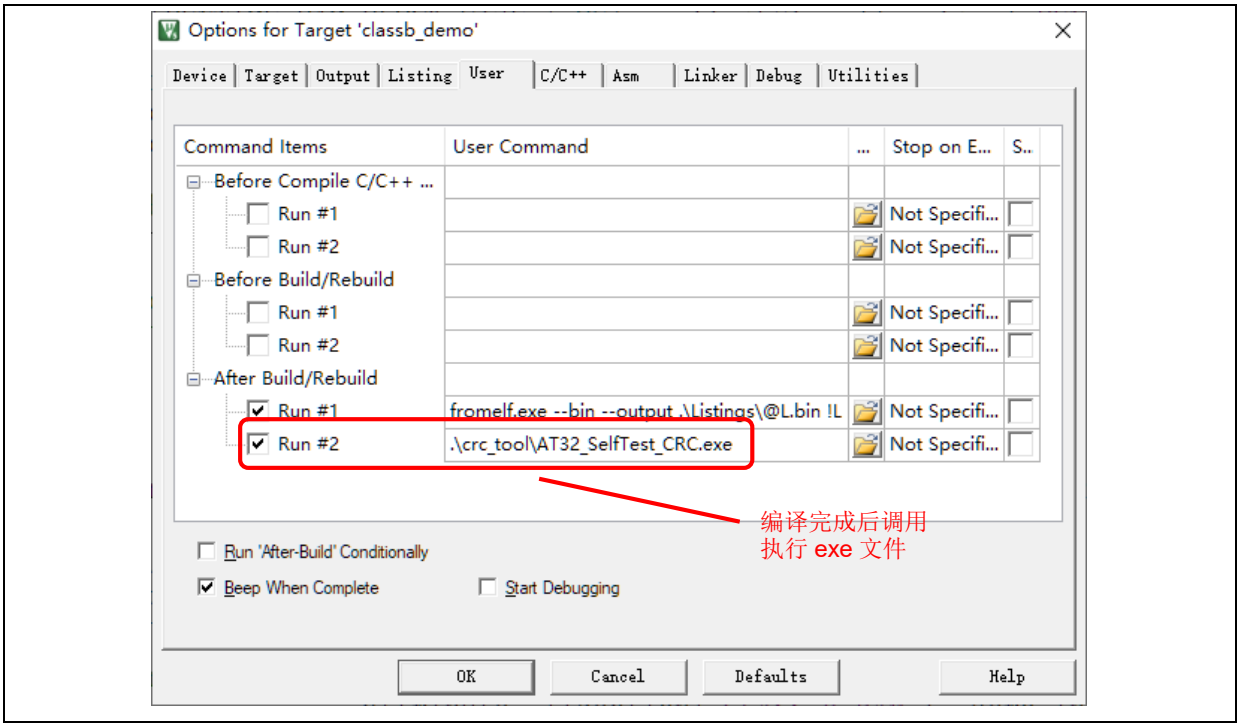
KEIL:

使用雅特力自己的工具AT32_SelfTest_CRC，将Keil生成的hex档添加CRC值另存为新的hex档（也支持bin档），程序代码中使用硬件CRC进行计算，描述设置CRC检测方法如下：

1) 将执行文件AT32_SelfTest_CRC.exe和参数文本SelfTest_CRC_Initparam.txt放在keil工程目录中的同一个文件夹，程序例程是放在crc_tool文件夹中

2) 工程中配置选择对应的文件夹路径调用AT32_SelfTest_CRC.exe，将KEIL编译生成的classb_demo.hex文件中的数据进行CRC计算，生成CRC chec_sum，合并到新的classb_demo_checked.hex文件中

图 8 keil 选项中添加 AT32_SelfTest_CRC.exe



SelfTest_CRC_Initparam.txt用于配置参数

1) input file：原始hex文件的名称和地址，地址为hex文件相对Kei工程文件的路径

2) output file：添加CRC的hex文件的名称和地址，地址为hex文件相对Kei工程文件的路径

3) start addr：计算CRC范围的开始地址

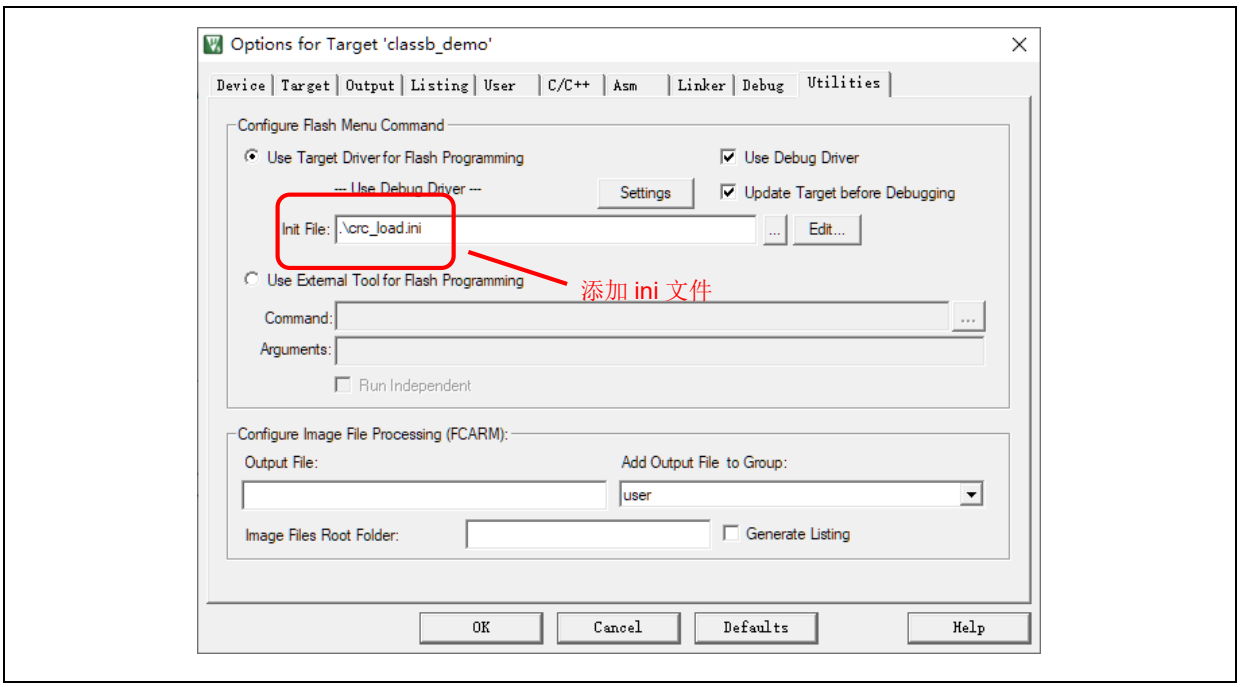
4) end addr：计算CRC范围结束，CRC checksum放置地址

5) fill blanks：CRC范围内原始hex文件的空白位置填充值，可以设置0xFF或者0x00

图 9 配置文件内容

```
input file = objects\classb_demo.hex
output file = objects\classb_demo_checked.hex
start addr = 0x08000000
end addr = 0x08006000
fill blanks = 0xFF
```

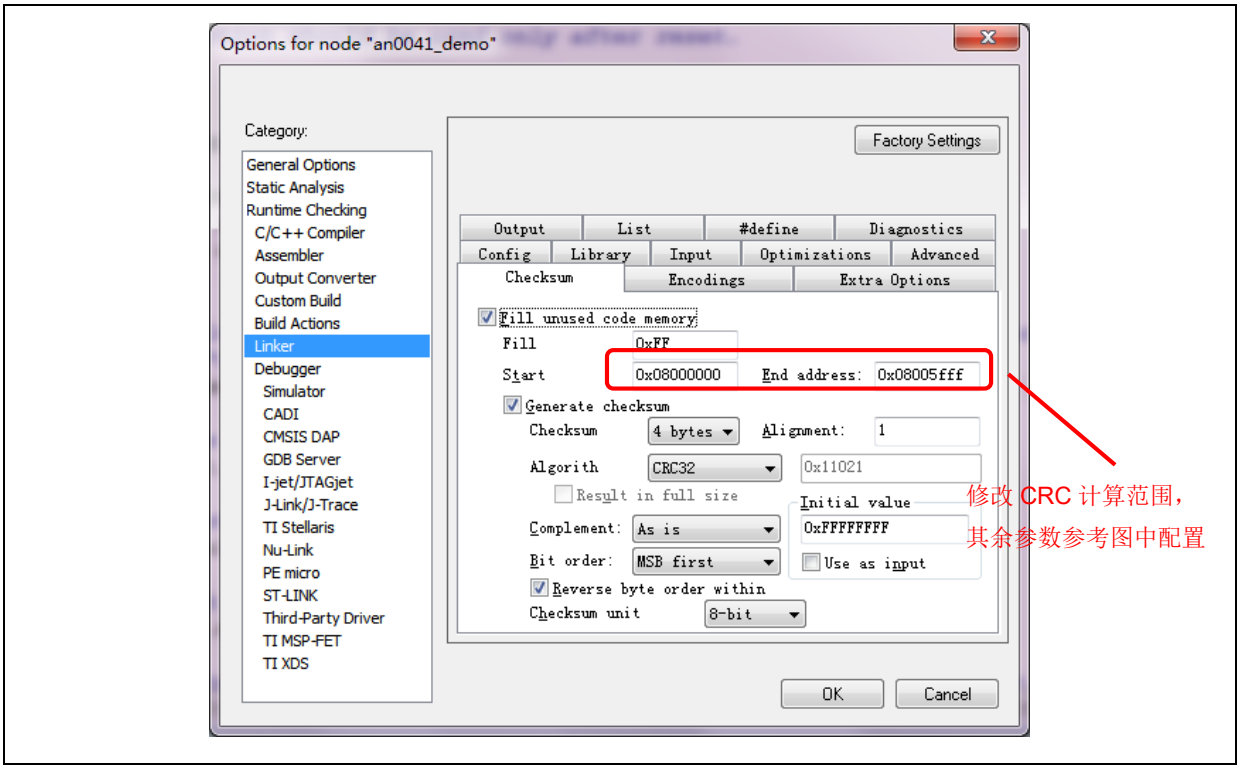
在下载或者调试中，都需要用最终生成的classb_demo_checked.hex代替原始的classb_demo.hex，所以在Keil配置选项中需添加crc_load.ini文件用于加载新的hex文件



IAR：

IAR配置选项支持CRC计算，只需要配置好参数，编译生成的文件就会自动将CRC check_sum值添加到选定Flash计算范围后面，Checksum参数配置是根据测试MCU型号的硬件CRC外设模块来确定，目前雅特力不同型号MCU的硬件CRC外设都相同，配置参数如下。

图 10 IAP 配置 CRC



AT32 IDE：

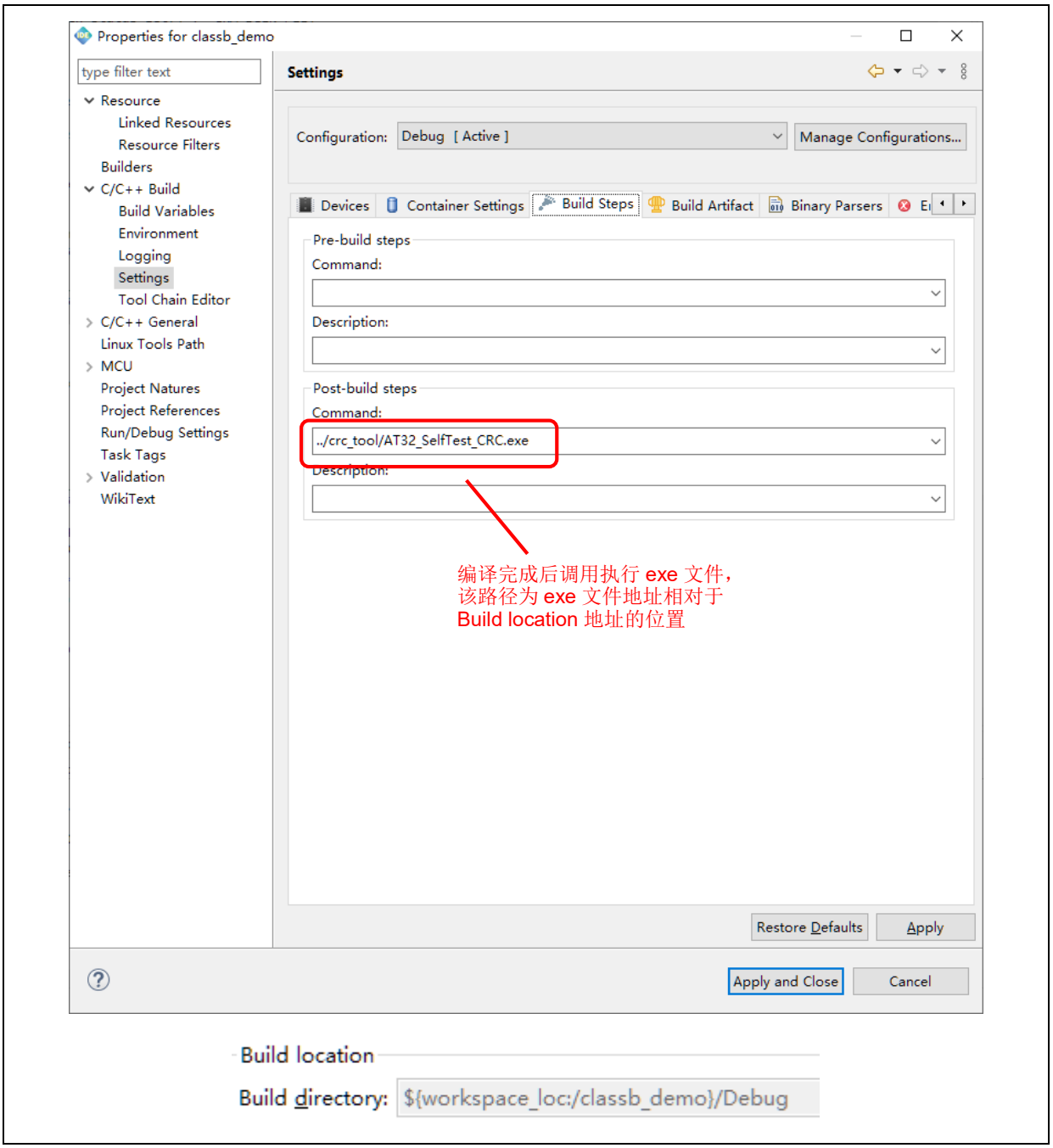
使用雅特力自己的工具AT32_SelfTest_CRC，将AT32 IDE生成的hex档添加CRC值另存为新的hex档（也支持bin档），程序代码中使用硬件CRC进行计算，描述设置CRC检测方法如下：

1) 将执行文件AT32_SelfTest_CRC.exe和参数文本SelfTest_CRC_Initparam.txt放在AT32 IDE工程目录中的同一个文件夹，程序例程是放在crc_tool文件夹中

2) 工程中配置选择对应的文件夹路径调用AT32_SelfTest_CRC.exe，将AT32 IDE编译生成的classb_demo.hex文件中的数据进行CRC计算，生成CRC chec_sum，合并到新的classb_demo_checked.hex文件中

SelfTest_CRC_Initparam.txt用于配置参数，具体参数内容参考上文中Keil章节的介绍。

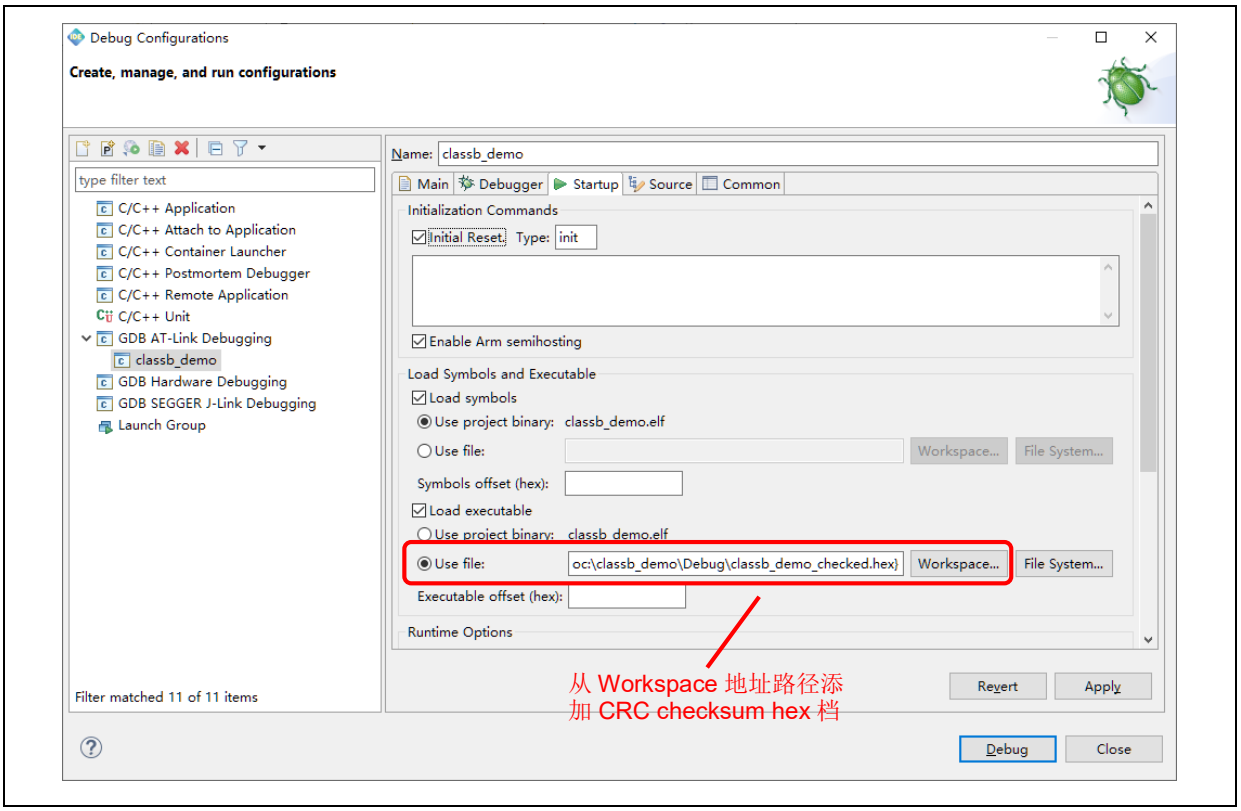
图 11 AT32 IDE 选项中添加 AT32_SelfTest_CRC.exe



在下载或者调试中，都需要用最终生成的classb_demo_checked.hex代替原始的classb_demo.hex，

所以在AT32 IDE配置选项中需配置加载新的hex文件，采用AT-Link调试器时配置如下图

图 12 AT32 IDE debug 选项加载新的有 CRC checksum 的 hex 档



注意：计算CRC值的Flash范围根据应用程序的实际大小确定，生成的CRC32 check sum存储地址需位于Flash空白区，否则可能覆盖修改到该存储地址的原始数据从而导致异常。

4.1.5 RAM 启动时检测

RAM自检是存储器检测的一部分，支持和不支持硬件SRAM奇偶校验功能的型号可以采用不同的方式检测。对于支持硬件SRAM奇偶校验功能的型号，如果使用硬件RAM检测的方式，需使能芯片的硬件RAM校验功能，该功能的开启推荐在量产阶段同代码烧录一起完成。前期开发设计阶段，可以通过ICP等工具配置开启。

支持硬件SRAM奇偶校验功能的型号

硬件SRAM校验功能开启后，因为上电SRAM数据是随机值，所以需要在程序启动阶段对SRAM数据进行初始化，对于Keil、IAR、AT32 IDE三种IDE，都是修改.s启动进行SRAM数据初始化，代码写法可能稍有差异，下图是Keil的.s启动文件添加SRAM初始化为全0x00的代码

图 13 Keil 启动文件添加 SRAM 初始化

```
174  IF :DEF:HW_RAM_CHECK
175                  LDR     R2, =0x20000000
176                  LDR     R3, =0x20018000
177                  LDR     R1, =0x00000000
178  __init_sram
179                  STR     r1, [r2]
180                  ADDS    r2, r2, #4
181                  CMP     r2, r3
182                  BCC     __init_sram
183  ENDIF
```

使用硬件SRAM检测功能，工程代码有两处宏定义修改：

1) at32_selftest_param.h文件中开启宏定义HW_RAM_CHECK

2) 工程汇编配置中开启宏定义HW_RAM_CHECK，下面分别是Keil、IAR、AT32 IDE配置截图

图 14 Keil 汇编配置开启 HW_RAM_CHECK

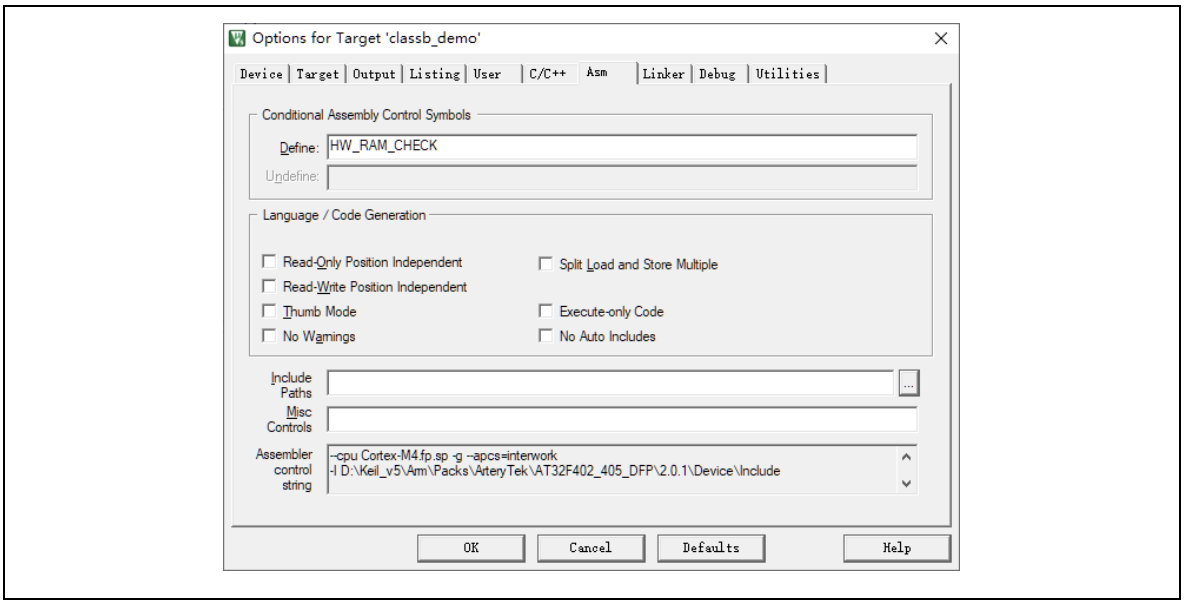


图 15 IAR 汇编配置开启 HW_RAM_CHECK

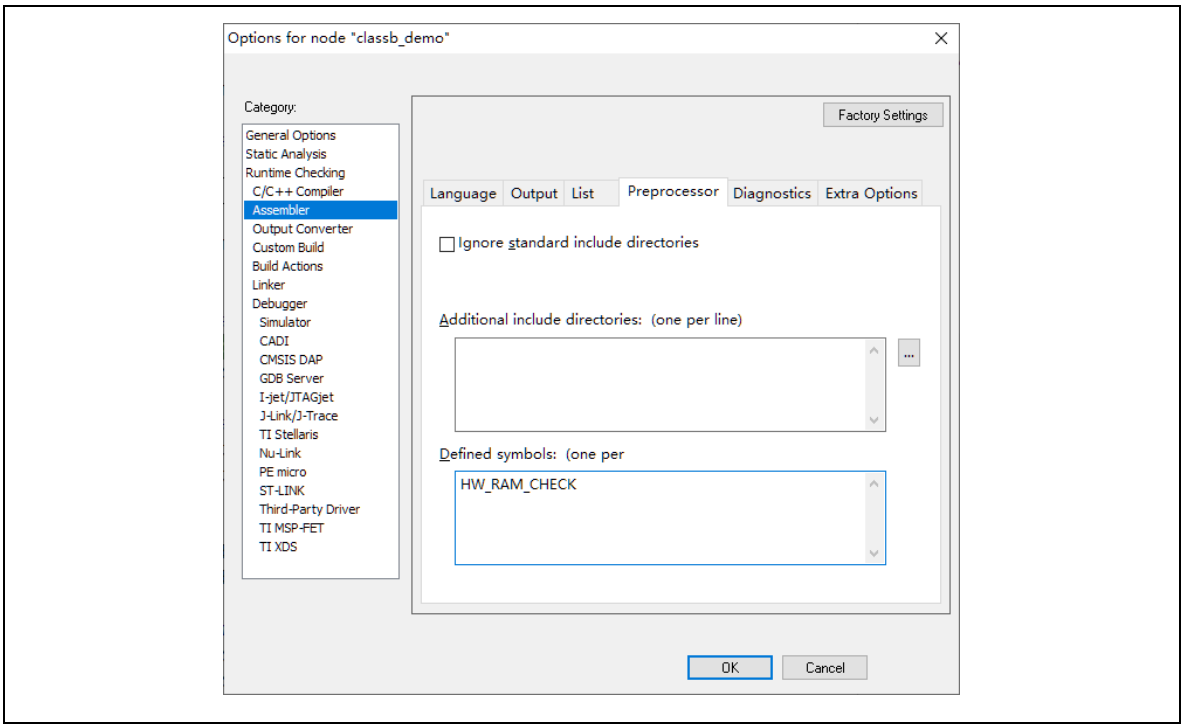
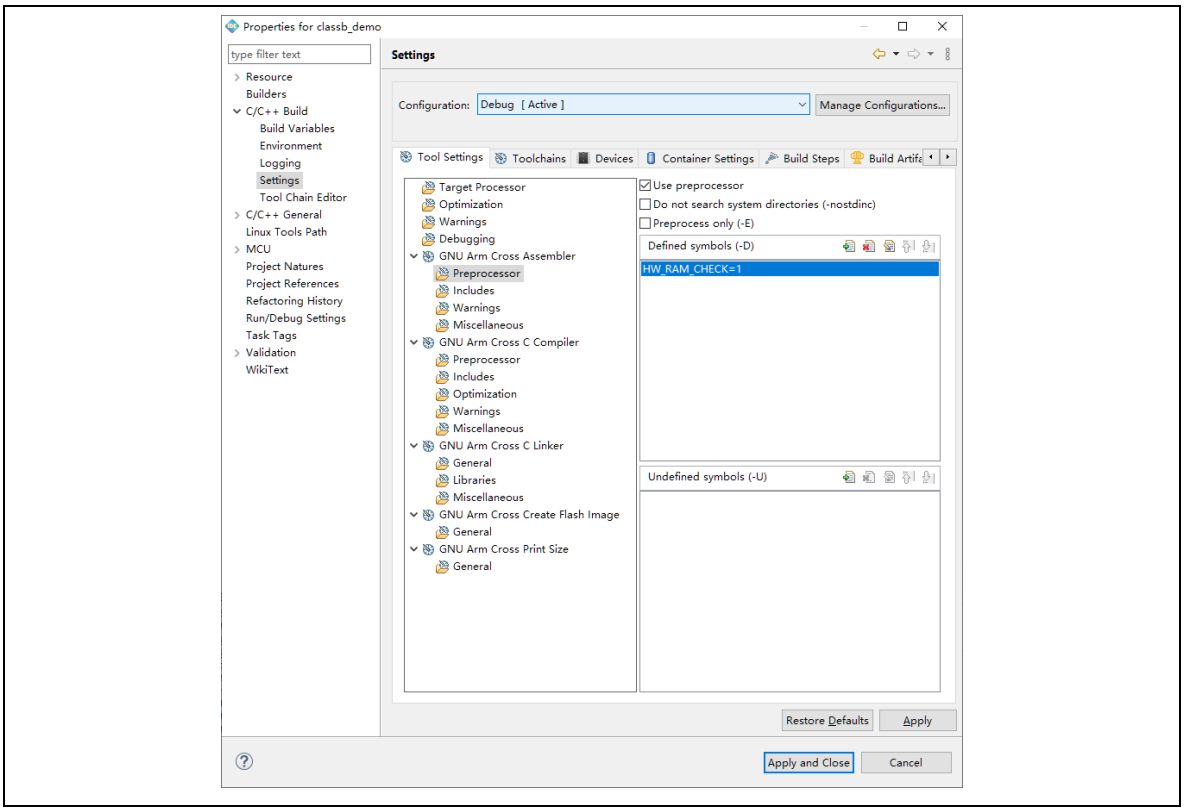


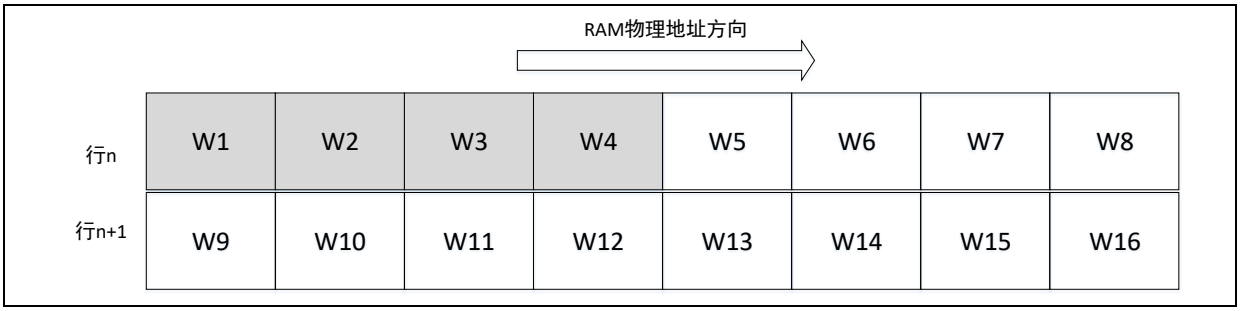
图 16 AT32 IDE 汇编配置开启 HW_RAM_CHECK



不支持硬件SRAM奇偶校验功能的型号

采用March C算法，用值0x55555555和0xAAAAAAAA逐字交替填充整个RAM并检查，基本物理单元是4字，下图单元格内的编号代表测试填充的顺序。

图 17 RAM 基本单元原理

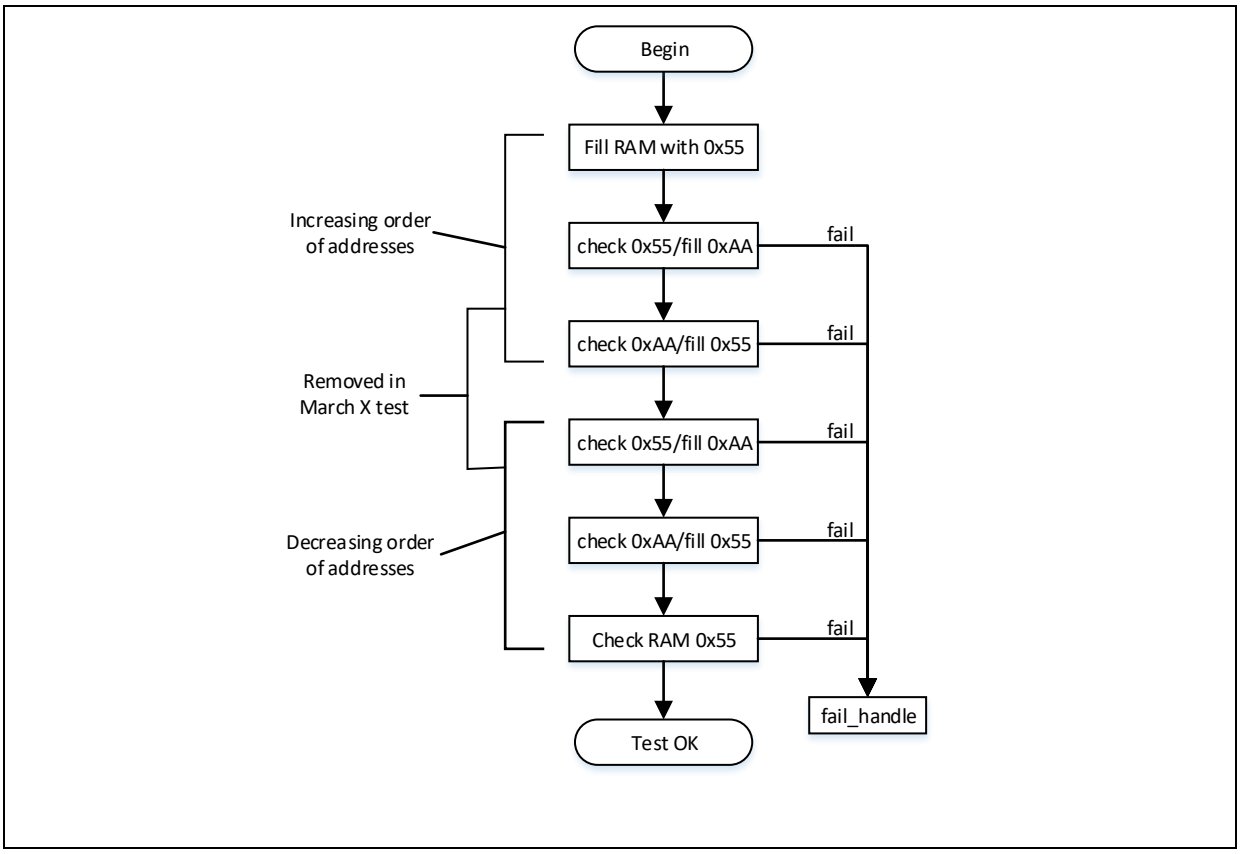


RAM采用March C算法测试时分6个步骤，前3个循环按照地址递增执行，后3个循环按照地址递减执行。测试步骤如下，如果采用March X算法则省略步骤3和4：

1. 全部范围写0x55555555，按照地址递增顺序执行
2. 检测是否全部范围为0x55555555，然后全部范围写0xAAAAAAAA，按照地址递增顺序执行
3. 检测是否全部范围为0xAAAAAAAA，然后全部范围写0x55555555，按照地址递增顺序执行
4. 检测是否全部范围为0x55555555，然后全部范围写0xAAAAAAAA，按照地址递减顺序执行
5. 检测是否全部范围为0xAAAAAAAA，然后全部范围写0x55555555，按照地址递减顺序执行
6. 检测是否全部范围为0x55555555，按照地址递减顺序执行

流程框图如下：

图 18 RAM 启动时检测流程



4.1.6 控制流启动时检测

控制流检测也属于程序计数器检测的一部分，启动阶段控制流检测主要分为了两个节点，因为RAM检测会导致所有变量被清除，包括定义为控制流检测相关的变量也会被清除，所以其中一个检测节点是在RAM检测之前，通过控制流变量值判断是否前面所有测试项都正确完成，另外一个节点是在RAM检测后，主要是运行阶段检测必须操作的流程的配置，比如CRC参考变量初始化、栈溢出pattern设置。

对于控制流检测，基本上每项检测模块都可以定义检测两层结构，其一是该项检测模块流程上是否正确调用(CALLER)，其二是该项检测模块是否被正确执行(CALLEE)，检测方法概述如下：

1. 定义两个变量指示控制流进度，设置初始值ctrl_flow_cnt为0x00000000，ctrl_flow_cnt_inv为0xFFFFFFFF，其初始状态互为取反的
2. 给每项测试模块定义两个固定数值分别表示CALLER和CALLEE，并赋予不同的值
3. 调用一项检测模块前，将ctrl_flow_cn增加CALLER的固定值，标示该模块已经调用
4. 进入对应检测模块内部，将ctrl_flow_cn增加CALLEE的固定值，标示该模块正在执行
5. 执行完对应检测模块内部，退出前ctrl_flow_cn_inv减少CALLEE的固定值，标示该模块执行正确
6. 完成对应检测模块，进入下一项检测模块前，将ctrl_flow_cn_inv减少CALLER的固定值，标示该模块调用正确
7. 检测ctrl_flow_cnt和ctrl_flow_cnt_inv是否仍互为取反的，如果是则表明对应检测模块流程上被正确调用，并且该检测模块被正确执行

4.2 运行时周期检测初始化

如果启动时的自检成功通过，并且标准初始化也完成了，运行时的周期自检必须在进入主循环之前进行初始化。该部分主要是一些后续自检会使用到的变量的初始化、中断中可能用到的数据的同步处理以及看门狗的配置等。

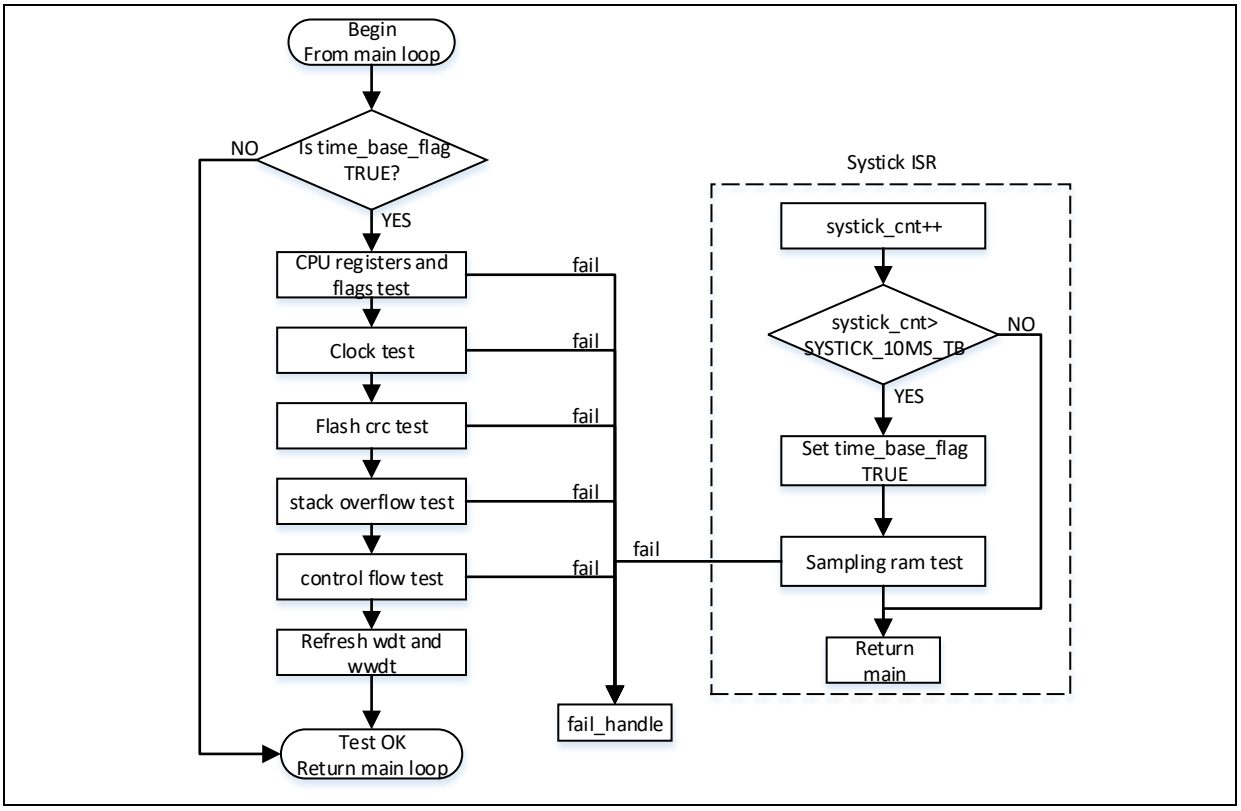
4.3 运行时周期检测流程

运行时的检测是以systick作为时基，进行周期性的检测。

运行时周期检测包括：

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

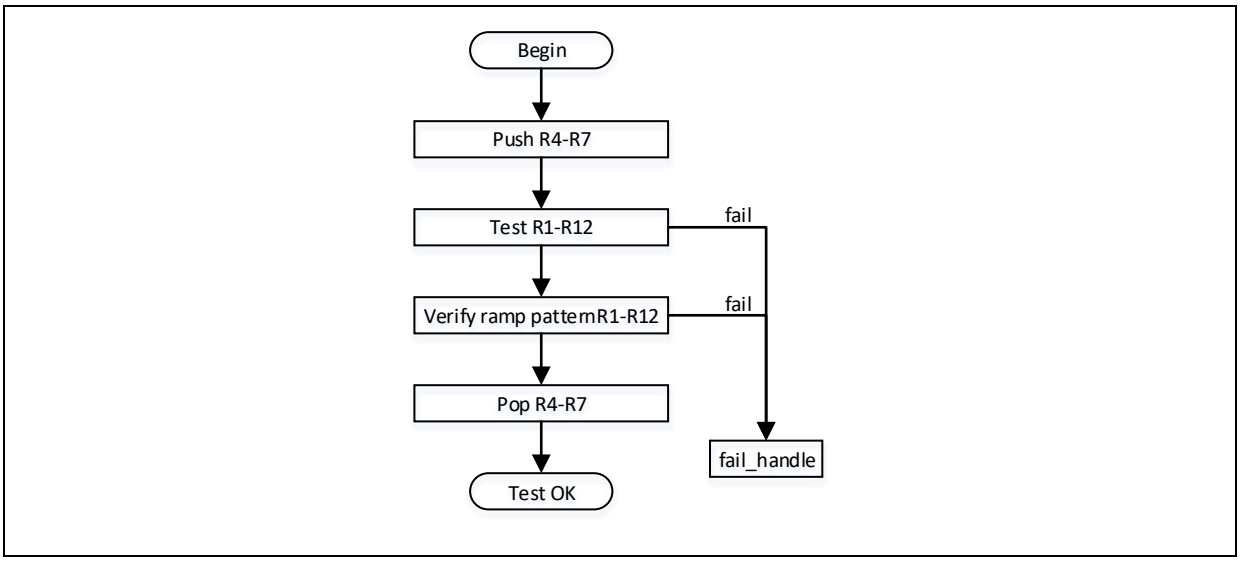
图 19 运行时周期自检及中断服务流程结构



4.3.1 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。

图 20 CPU 运行时检测流程

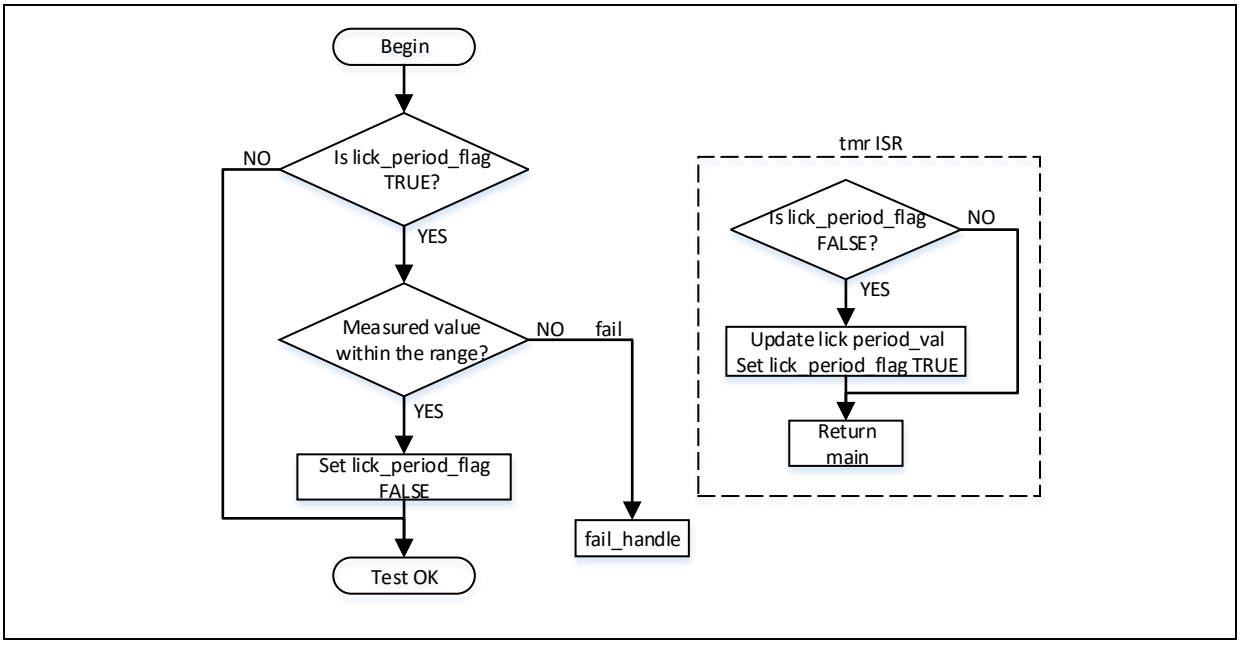


4.3.2 系统时钟运行时检测

运行时系统时钟的检测跟启动时时钟检测原理相同。因为运行时的时钟检测会一直循环执行，对某些用户应用，如果采用定时器捕获LICK边沿的方式，可能中断频率太高会有影响，所以也可以采用其他方式进行测试，测试原理不变仍然是通过内部低速时钟源（LICK）和系统时钟的交叉测量结果来验证，AT32F403A和AT32F415例程分别采用了两种不同的测试方法，供用户参考。

- 在AT32F403A例程中，跟启动时时钟检测方法相同，仍然采用专用的定时器TMR输入捕获LICK边沿频率的方式进行测试，下图是采用定时器捕获LICK测试方式的流程。

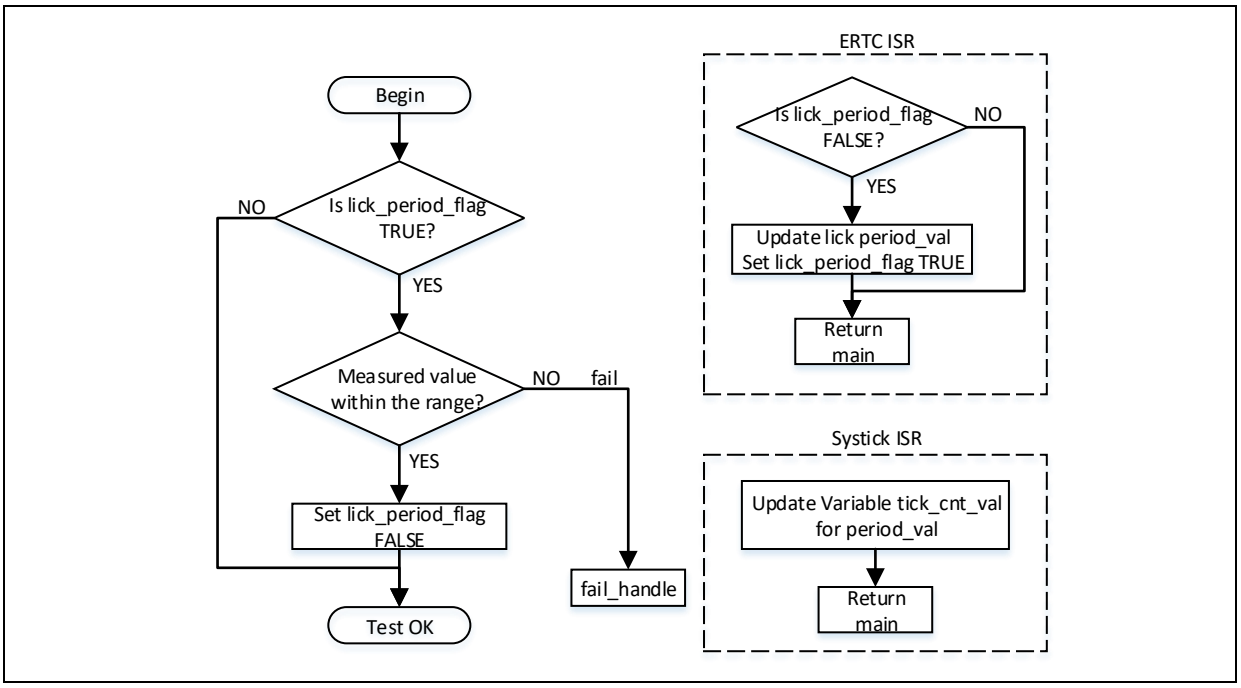
图 21 系统时钟运行时定时器捕获 LICK 检测方式流程



- 在AT32F415的例程中，使用ERTC（时钟源为LICK）和Systick定时器（时钟源为系统时钟）之间交叉测量，ERTC按照数据手册上LICK典型值作为时钟源配置秒中断参数，Systick定时器中断配置为1毫秒，Systick中断计数变量tick_cnt_val累加，ERTC秒中断时获得Systick的计数变量数值，以此计算得到LICK实际测试值，如果在规格范围内的最大和最小值之间，测试通过。下

图是采用ERTC获取Systick变量值测试方式的流程图。

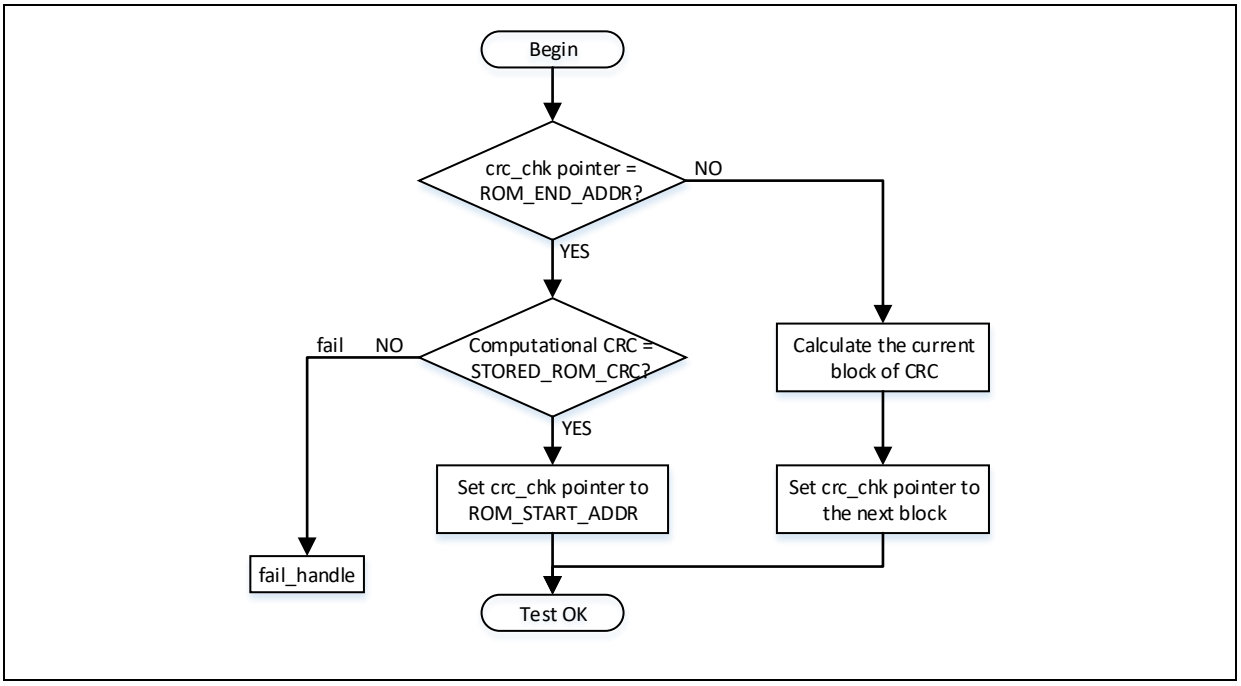
图 22 系统时钟运行时 ERTC 获取 Systick 值检测方式流程



4.3.3 Flash CRC 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，如果一次计算检测全部范围CRC可能耗时过长，影响正常应用部分的执行，所以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，再进行CRC值比较，如果不一致则测试失败。

图 23 Flash CRC 运行时检测流程

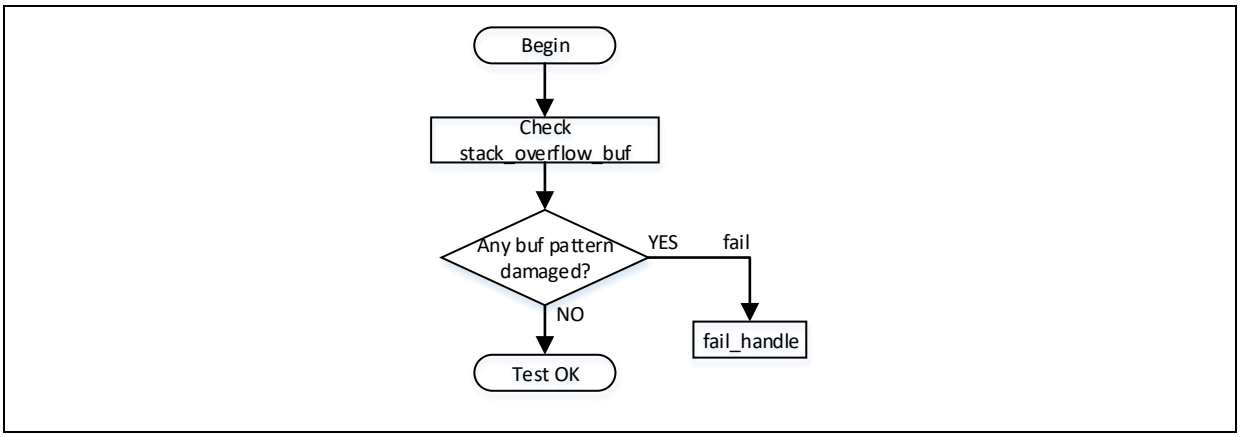


4.3.4 栈边界运行时溢出检测

可验证寻址和数据路径测试相关，定义特殊固定值的Magic Pattern数组，放置在栈区域最底部地址，通过运行中检测Magic Pattern数组完整性来判断栈是否溢出。如果原始Pattern被破坏，则表明栈溢出测试失败，调用故障安全程序。

这一区域根据设备及应用有不同的配置。用户必须为堆栈定义足够的区域，并保证pattern正确放置。

图 24 堆栈边界溢出运行时检测流程



4.3.5 RAM 运行时检测

对于不支持硬件SRAM奇偶校验功能的型号，运行时的RAM自检是在systick中断函数中进行的。测试范围可以根据用户实际应用需求调整，通过修改代码中宏定义RUNTIME_RAM_START_ADDR和RUNTIME_RAM_END_ADDR来完成，需注意的是因为测试包括了测试区域前后相邻的字，所以测试范围前后要保留适当余量，不要覆盖临时保存数据的缓冲块(buffer block)和溢出芯片RAM范围。测试流程方法概要描述如下：

- 测试根据systick时基分批次进行，每次测试按照CLASS B部分4个字的区域偏移，为保障耦合故障覆盖率，每次测试的实际内存块还包括测试区域前后各1个相邻字，总共6个字。
- 首先将待测试内存块(RAM block)的数据存储到专门用于测试过程中临时保存数据的缓冲块(buffer block)
- 然后跟启动时检测RAM类似，对测试内存块(RAM block)采用March C算法测试
- 测试完成后将缓冲块(buffer block)中保存的数据恢复至测试内存块(RAM block)

下图描述了故障耦合的基本原理，图中数据编号表示操作先后顺序。

图 25 局部 RAM 检测故障耦合加扰模式原理

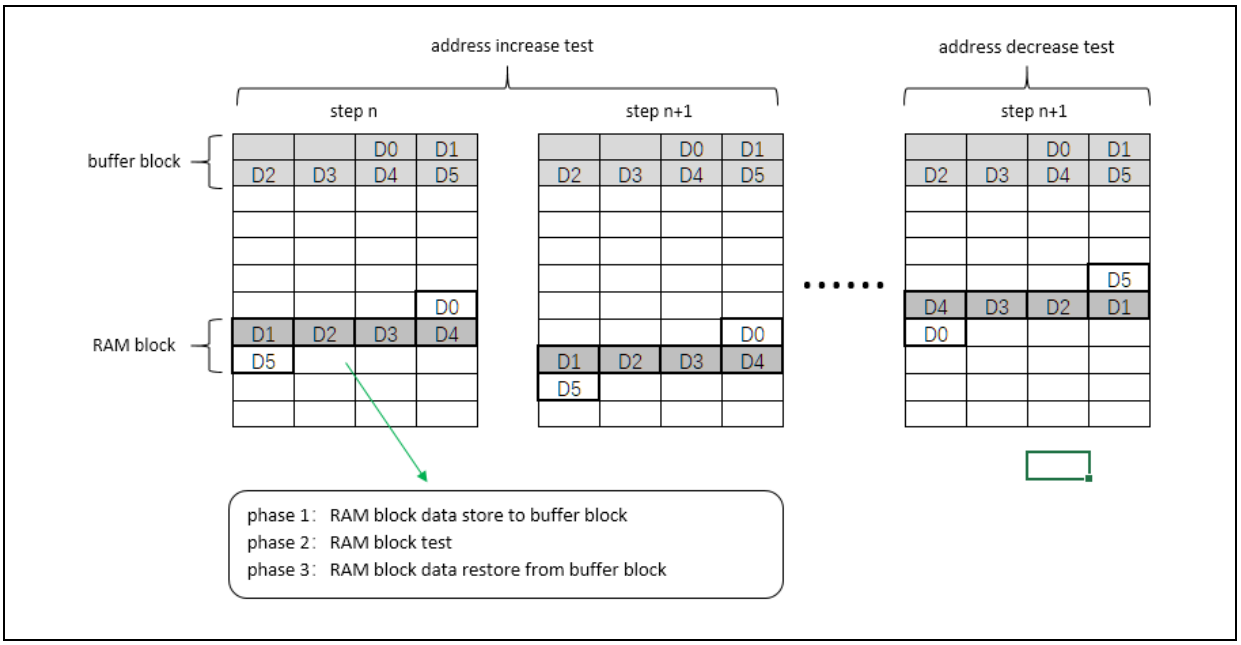
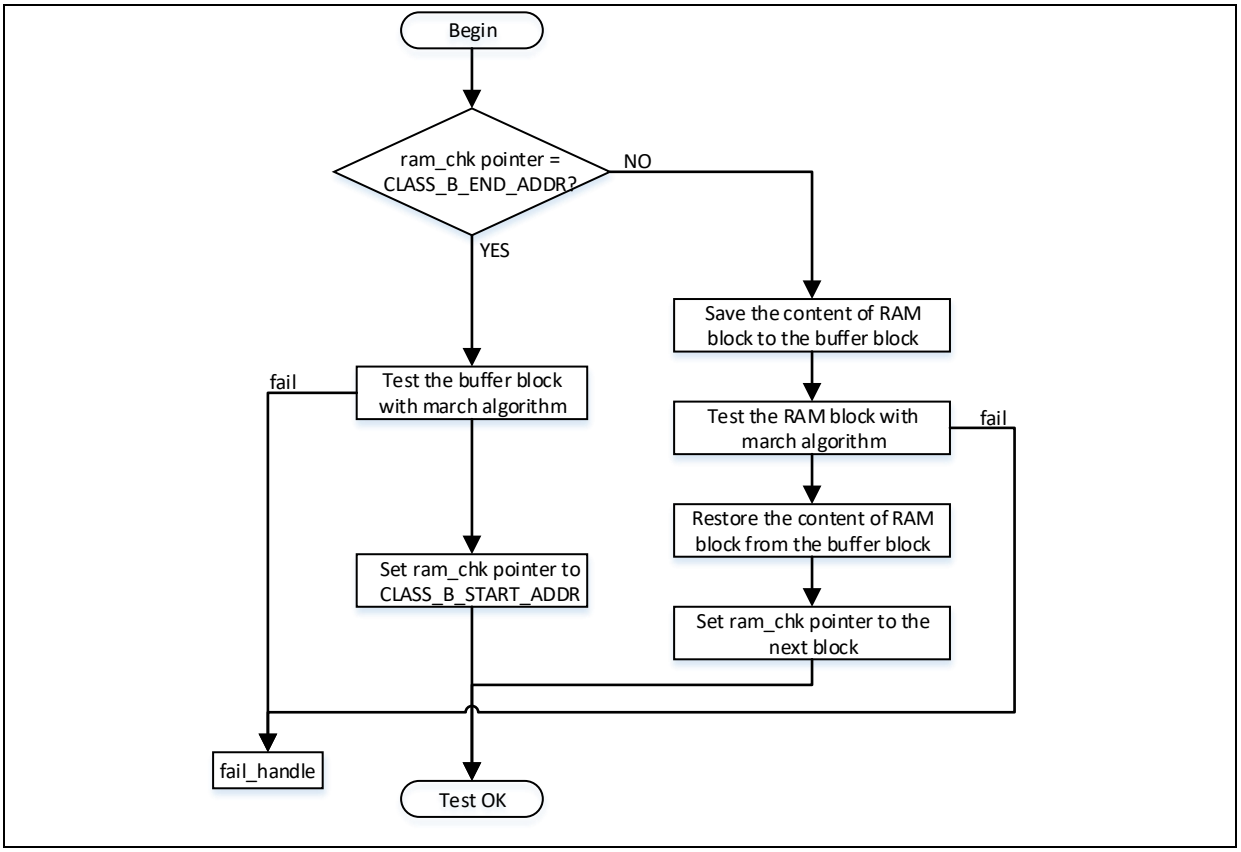


图 26 局部 RAM 运行时检测流程



4.3.6 看门狗运行时刷新

作为程序计数器检测的一部分，运行时需要定期喂狗保证系统正常运行，看门狗的刷新部分放置在每次检测最后部分。

注意：运行时的看门狗超时时间，根据用户程序中刷新喂狗代码的调用频率配置，软件库源代码通常采用的是10ms调用一次，所以源代码中看门狗超时时间配置大于10ms。

5 程序注意事项

用户使用中可能会根据实际情况对CLASSB代码进行修改，如果改动不符合要求可能导致检测失败，本章节列举了一些常见的注意事项。

5.1 程序中自定义参数修改

系统主频根据实际应用修改，函数为selftest_system_clock_config()。

其余配置参数定义在at32_selftest_param.h文件以及各自集成开发环境的链接或者分散加载文件中，根据实际应用进行修改，下面分别对Keil、IAR、AT32 IDE进行说明。

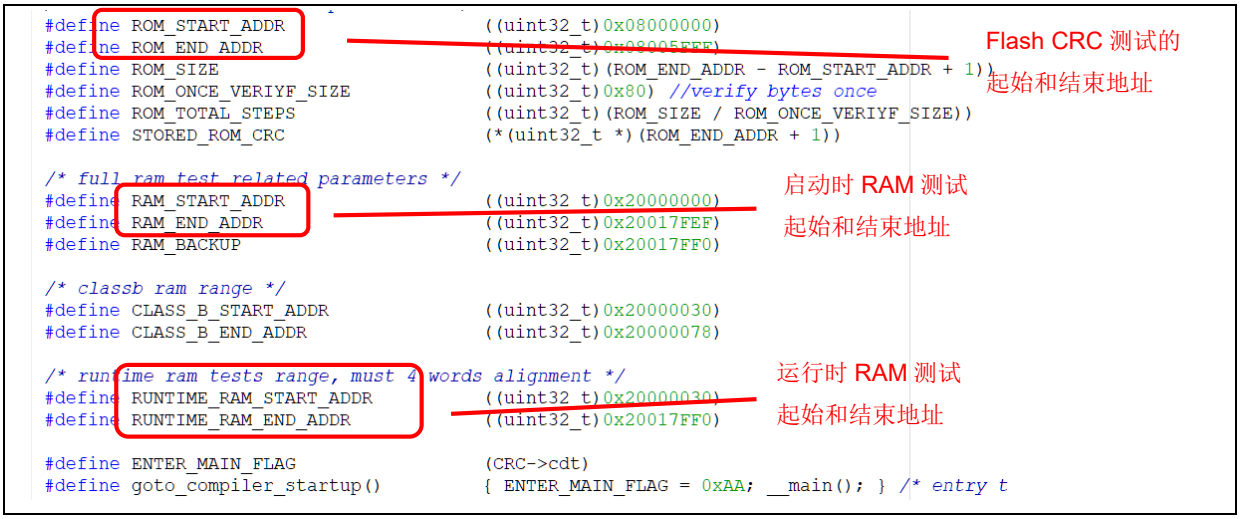
at32_selftest_param.h文件中各个集成开发环境通用部分如下：

图 27 通用可修改参数

at32_selftest_param.h文件中跟各个集成开发环境相关写法不一致的部分如下

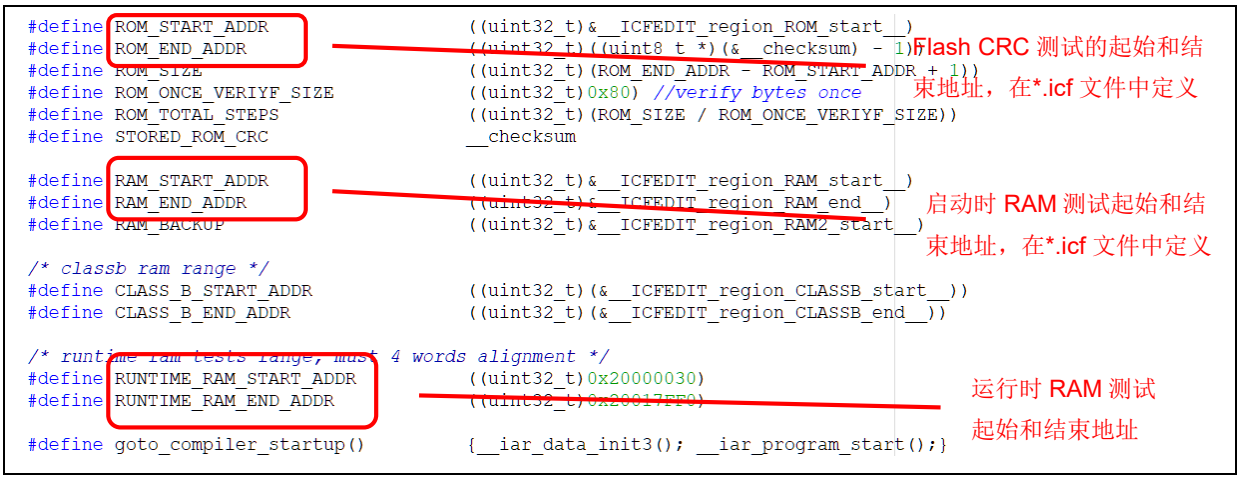
Keil：

图 28 Keil 可修改参数



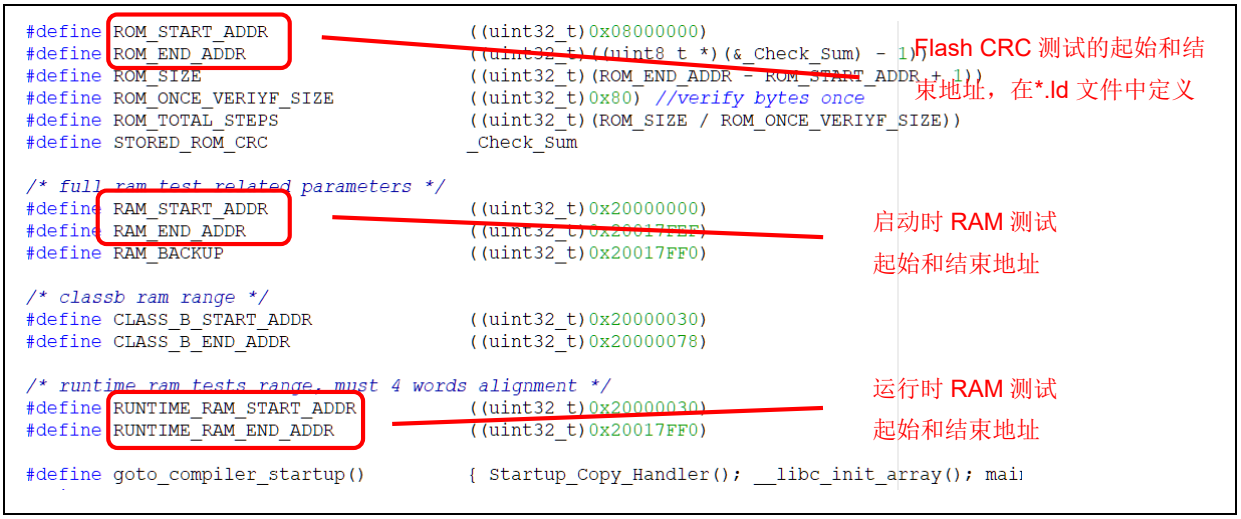
IAR：

图 29 IAR 可修改参数



AT32 IDE

图 30 AT32 IDE 可修改参数



5.2 Flash CRC 检测范围设置

因为在Flash CRC运行时检测是分段进行的，所以设置的总大小范围需要跟代码中的分段大小宏定义ROM_ONCE_VERIYF_SIZE成整数倍对齐，否则可能导致最后计算CRC失败。

5.3 RAM 检测范围设置

因为在运行时检测是每次4个字，所以检测范围也需要保证4字对齐，并且因为检测范围前后还需2个字的空隙，所以设置范围时要防止覆盖到测试RAM时临时保存数据的缓冲块(buffer block)或者溢出芯片RAM范围，导致在RAM测试时程序出错。

5.4 编译器影响

1. 编译器的优化不仅会导致程序分析调试困难，还可能导致程序出现非预期的结果，所以强烈建议在使用中针对CLASSB部分代码不要进行优化。
2. 当使用不同版本的编译器时，程序可能有差异。新版CLASS软件库对于不同型号MCU的工程，

为方便使用不同编译器版本的用户移植，可能使用了不同的编译器版本。

6 软件库说明

6.1 软件库下载

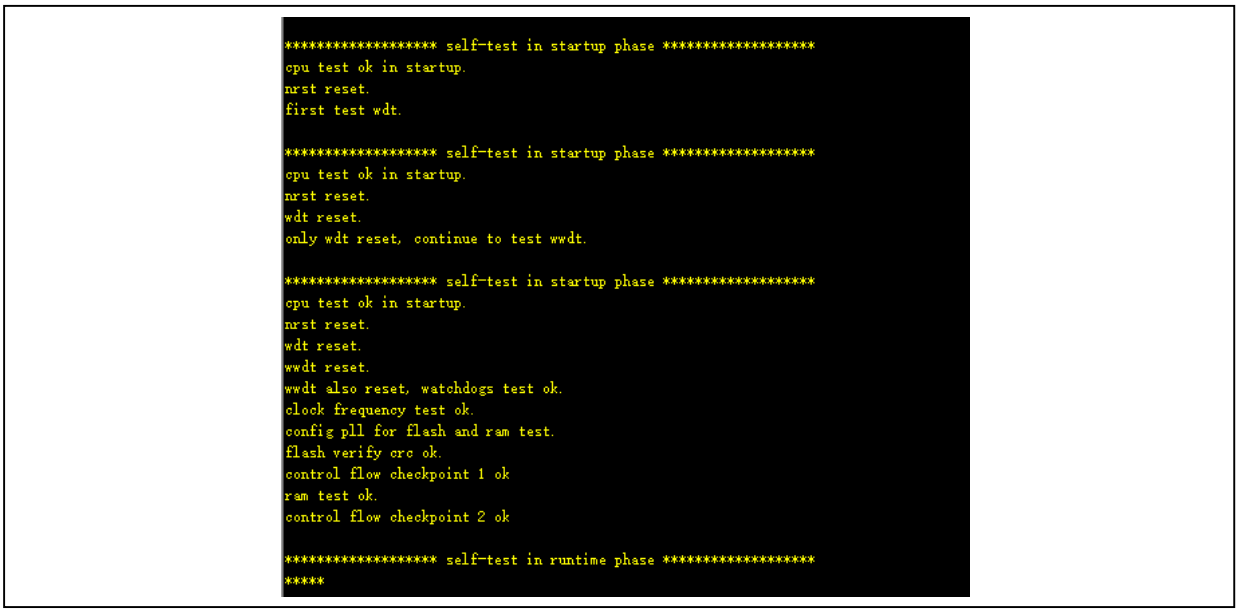
各个已支持型号对应的软件库源代码开放于Example Code中，用户可以通过官网下载，官网中Example Code内容的链接地址为https://www.arterytek.com/cn/support/index.jsp?index=2，可搜索关键字“60730”快速查找已支持型号MCU的软件库，例如AT32F403A的软件库文件为“AT32F403A 407 IEC 60730 CLASSB LIB”。

6.2 软件库演示

下面以SC0127 AT32F403A 407 IEC 60730 CLASSB LIB为例，AT32F403A安全库的例程运行环境如下：

- KEIL工程为V5.36版本，IAR工程为V8.2版本，AT32 IDE工程为V1.0.09版本。
- Demo运行基于F403A芯片，运行在AT-START-F403A开发板。
- 工程路径为：utilities\classb_demo

图 31 运行打印信息



7 版本历史

表 3 文档版本历史

日期	版本	变更
2022.3.15	2.0.0	最初版本
2022.11.4	2.1.0	1.更新Flash启动时检测分别采用两种不同的CRC方式 2.添加“注意事项”章节
2023.3.29	2.1.1	1.更新时钟检测的描述 2.更新局部RAM自检的描述
2024.2.22	2.2.0	1.更新CPU测试的寄存器描述 2.更新RAM测试方式和运行阶段RAM测试的范围设置描述
2024.3.11	2.3.0	1.添加AT32 IDE编译环境的介绍 2.增加自定义参数修改描述
2024.4.2	2.3.1	1.添加例程代码的相关说明 2.更新时钟检测的描述
2024.5.13	2.3.2	1.添加Keil AC6编译环境的介绍 2.更新RAM自检软件和硬件两种方式的描述 3.更新部分描述内容

重要通知 – 请仔细阅读

买方自行负责对本文所述雅特力产品和服务的选择和使用，雅特力概不承担与选择或使用本文所述雅特力产品和服务相关的任何责任。

无论之前是否有过任何形式的表示，本文档不以任何方式对任何知识产权进行任何明示或默示的授权或许可。如果本文档任何部分涉及任何第三方产品或服务，不应被视为雅特力授权使用此类第三方产品或服务，或许可其中的任何知识产权，或者被视为涉及以任何方式使用任何此类第三方产品或服务或其中任何知识产权的保证。

除非在雅特力的销售条款中另有说明，否则，雅特力对雅特力产品的使用和/或销售不做任何明示或默示的保证，包括但不限于有关适销性、适合特定用途（及其依据任何司法管辖区的法律的对应情况），或侵犯任何专利、版权或其他知识产权的默示保证。

雅特力产品并非设计或专门用于下列用途的产品：（A）对安全性有特别要求的应用，例如：生命支持、主动植入设备或对产品功能安全有要求的系统；（B）航空应用；（C）航天应用或航天环境；（D）武器，且/或（E）其他可能导致人身伤害、死亡及财产损害的应用。如果采购商擅自将其用于前述应用，即使采购商向雅特力发出了书面通知，风险及法律责任仍将由采购商单独承担，且采购商应独力负责在前述应用中满足所有法律和法规要求。

经销的雅特力产品如有不同于本文档中提出的声明和/或技术特点的规定，将立即导致雅特力针对本文所述雅特力产品或服务授予的任何保证失效，并且不应以任何形式造成或扩大雅特力的任何责任。