XHSC 小华半导体
XIAOHUA SEMICONDUCTOR

32 位微控制器

IEC60730_Class_B_STL

应用笔记

Rev1.11　2026 年 08 月

适用对象

产品系列	产品型号	产品系列	产品型号	产品系列	产品型号
HC32L130	ALL	**HC32L136**	ALL	**HC32F030**	ALL
HC32L176	ALL	**HC32L170**	ALL	**HC32F176**	ALL
HC32F170	ALL	**HC32L196**	ALL	**HC32L190**	ALL
HC32F196	ALL	**HC32F190**	ALL	**HC32F052**	ALL
HC32L186	ALL	**HC32L180**	ALL	**HC32F155**	ALL
HC32F115	ALL	**HC32F420**	ALL	**HC32F460**	ALL
HC32D391	ALL	**HC32F451**	ALL	**HC32F452**	ALL
HC32F448	ALL	**HC32M441**	ALL	**HC32F472**	ALL
HC32F4A0	ALL	**HC32F467**	ALL	**HC32F4A8**	ALL
HC32M413	ALL	**HC32F431**	ALL	**HC32M458**	ALL

声　明

★ 小华半导体有限公司（以下简称：“XHSC”）保留随时更改、更正、增强、修改小华半导体产品和/或本文档的权利，恕不另行通知。用户可在下单前获取最新相关信息。XHSC 产品依据购销基本合同中载明的销售条款和条件进行销售。

★ 客户应针对您的应用选择合适的 XHSC 产品，并设计、验证和测试您的应用，以确保您的应用满足相应标准以及任何安全、安保或其它要求。客户应对此独自承担全部责任。

★ XHSC 在此确认未以明示或暗示方式授予任何知识产权许可。

★ XHSC 产品的转售，若其条款与此处规定不同，XHSC 对此类产品的任何保修承诺无效。

★ 任何带有“®”或“™”标识的图形或字样是 XHSC 的商标。所有其他在 XHSC 产品上显示的产品或服务名称均为其各自所有者的财产。

★ 本通知中的信息取代并替换先前版本中的信息。

目　录

1 概述

本应用笔记描述如何使用和实现提供的库函数功能，并通过测试工程演示如何使用测试库。用户可参考该工程，将安全自检功能集成到用户工程。

2 IEC60730 测试条目

IEC 60730 标准包括三类：

- A 类：不用于确保设备的安全性
- B 类：防止受控设备的不安全运行
- C 类：防止特殊危害

IEC60730 自检测试库（Self Test Library，STL）按照 IEC60730-1：2022 附录 H 中表 H.11.12.7 软件类 B 要求实现自检功能。IEC 60730-1 的附录 H 标准虽然没有包含终端应用，但通常也需要进行看门狗和堆栈溢出检测。

IEC60730 自检测试库测试功能如表 2-1 所示。

表 2-1 IEC60730 测试条目

项目	测试项目	故障/错误	测试方法	STL是否支持
1.1	CPU寄存器	固定故障	H. 2.16.6 H. 2.19.6	是
1.3	PC寄存器	固定故障	H. 2.16.5 H. 2.16.6 H. 2.18.10.4	是
2	中断	无中断或中断过于频繁	H. 2.16.5 H. 2.18.10.4	是
3	时钟	错误频率	H. 2.18.10.1	是
4.1	不可变存储器	所有单比特故障	H. 2.19.8.2 H. 2.19.4.2	是
4.2	可变存储器	DC故障	H. 2.19.6	是
7.2.1	模拟AD转换器	H.27指定的故障条件	H. 2.18.13	是
-	看门狗	—	—	是
-	堆栈	—	—	是

3 IEC60730 Class B 库函数

这一章概述各种功能库函数接口和测试方法。

自检测试包含两个阶段：

a、启动时测试（Startup Self Test），只在上电启动过程中，测试一次。

b、运行时测试（Run Time Self Test），主程序中持续进行测试。

测试库实现接口如表 3-1 所示：

表 3-1 库函数列表

API	文件	说明	ROM用量[注5]（字节）	RAM用量[注5]（字节）	执行时间[注5]（微秒）
STL_DelayMS[注4]	stl_common.c/stl_utility.c	延迟功能	-	-	-
STL_DelayUS[注4]		延迟功能	-	-	-
STL_PrintfInit[注4]	stl_debug.c/stl_utility.c	库调试打印初始化(optional)	-	-	-
STL_SafetyFailure[注4]	stl_debug.c/stl_bsp_conf.c	测试失败处理	-	-	-
STL_CpuTestStartup[注2]	**stl_test_cpu_startup_xxx.s[注1]/ stl_test_cpu_cm4_startup_xxx.s[注1]**	**CPU寄存器启动时测试**	**650**	**-**	**14.5**
STL_CpuTestRuntime[注2]	**stl_test _cpu _runtime_xxx.s[注1]**	**CPU寄存器运行时测试**	**348**	**-**	**9**
STL_PcTest[注2]	**stl_test _pc _xxx.s[注1]**	**PC寄存器启动/运行时测试**	**148**	**-**	**4**
STL_Crc32Init	stl_sw_crc32.c	初始化CRC32初始值	-	-	-
STL_GetCrc32		获取CRC32当前值	-	-	-
STL_StartupCalculateCRC32Value		获取CRC32当前值（启动测试）	-	-	-
STL_CalculateCRC32Value		计算指定空间CRC32校验值（运行时测	-	-	-

API	文件	说明	ROM用量[注5]（字节）	RAM用量[注5]（字节）	执行时间[注5]（微秒）
		试）			
STL_FlashStartupParaInit[注4]	test_impl_flash.c	Flash启动测试参数初始化	-	-	-
STL_FlashRuntimeParaInit[注4]		Flash运行测试参数初始化	-	-	-
STL_FlashStartupTest[注2]	**stl_test _flash.c**	**Flash启动时测试**	**48**	**44**	**142000**
STL_FlashRuntimeInit[注2]		**Flash运行时测试初始化**	**16**	**36**	**-**
STL_FlashRuntimeTest[注2]		**Flash运行时测试**	**136**	**40**	**6**
STL_RamRuntimeParaInit[注4]	test_impl_ram.c	RAM运行测试参数初始化			
STL_FullRamTestStartup[注2]	**stl_test _full_ram_xxx.s[注1]/ stl_test _full_ram_startup_xxx.s[注1]**	**RAM启动时测试**	**216**	**-**	**10700**
STL_StackRuntimeInit[注2]	**stl_test_ram_runtime.c**	**Stack运行时测试初始化**	**22**	**16**	**-**
STL_StackRuntimeTest[注2]		**Stack运行时测试**	**48**	**16**	**1.5**
STL_RamRuntimeInit[注2]		**RAM March C运行时测试初始化**	**18**	**20**	**-**
STL_RamRuntimeTest[注2]		**RAM March C运行时测试**	**558**	**28**	**27**
System_Clock_Init/ BSP_CLK_Init[注4]	test_impl_clk.c/ev_hc32xxxx_lqfpxxx.c	系统时钟配置	-	-	-
STL_ClkParaInit[注4]	test_impl_clk.c	时钟和中断测试参数初始化	-	-	-
STL_ClockTestInit[注4]		时钟和中断测试初始化	-	-	-
SysClktimer_IRQHandler[注3][注4]		系统时钟定时器中断处理	-	-	-
LrcClktimer_IRQHandler[注3][注4]		低速时钟定时器中断处理	-	-	-
STL_SysTimerIrq[注2]	**stl_test _clock.c**	**时钟和中断测试计数**	**16**	**8**	**1**
STL_ClockTest[注2]		**时钟和中断测试处理**	**80**	**17**	**2**
STL_ClockStartupTest[注2]		**时钟启动时测试**	**36**	**40**	**500**
STL_ClockRuntimeTest[注2]		**时钟和中断运行时测试**	**14**	**8**	**1**
STL_NonIntTestRuntime[注2]		**无中断测试**	**18**	**12**	**1**

API	文件	说明	ROM用量[注5]（字节）	RAM用量[注5]（字节）	执行时间[注5]（微秒）
STL_AdcTestParaInit[注4]	test_impl_adc.c	ADC启动测试参数初始化	-	-	-
STL_AdcstartupInit[注4]/ STL_AdcTestStartupInit[注4]		ADC启动测试初始化	-	-	-
STL_AdcstartupDeInit[注4] STL_AdcTestStartupDeInit[注4]		ADC去初始化	-	-	-
STL_AdcTestStartup[注2]	**stl_test _adc.c**	**ADC启动测试**	**126**	**48**	**14.7**
STL_WdtStartupParaInit[注4]	test_impl_wdt.c	看门狗启动测试参数初始化	-	-	-
STL_WdtStartupTestInit[注4]		看门狗启动测试初始化	-	-	-
STL_WdtTestStart[注4]		看门狗启动	-	-	-
STL_WdtRuntimeInit[注4]		看门狗运行初始化	-	-	-
STL_WdtRuntimeFeed[注4]		看门狗喂狗	-	-	-
STL_WdtStartupTest[注2]	**stl_test _wdt.c**	**看门狗启动测试**	**108**	**25**	**55**

注 *1*： *xxx:*表示指定的编译器：*iar = IAR EWARM* 编译器；*keil = Keil µVision* 编译器；

注 *2*： 该列表加粗字体行表示认证库所包含文件和函数；

注 *3*： 该中断函数名称可以自由定义，在不同样例工程中可能存在不同名称；

注 *4*： 此函数需要用户根据不同芯片驱动库的要求做相应调整实现；

注 *5*： 以 *HC32L18x* 例程为参考：系统时钟 *48MHz*；*RAM 0 waitcycle*；*Flash 1 waitcycle*；*Flash Cache*：*None*。

3.1 CPU 寄存器测试

Cortex-M0+和 Cortex-M4 CPU 的寄存器组中，包含 13 个 32 位通用目的寄存器，以及多个特殊功能寄存器。所测寄存器如表 3-2 和表 3-3 所示。

表 3-2 Cortex-M0+ CPU 寄存器列表

寄存器	测试位
R0-R12	[31:0]
R13 (SP_main, SP_process)	[31:2]
R14 (LR)	[31:0]
APSR	[31:28]
CONTROL	[1]
PRIMASK	[0]

表 3-3 Cortex-M4 CPU 寄存器列表

寄存器	测试位
R0-R12	[31:0]
R13 (SP_main, SP_process)	[31:2]
R14 (LR)	[31:0]
APSR	[31:27]
CONTROL	[1]
PRIMASK	[0]
FAULTMASK	[0]
BASEPRI	[7:4]

3.1.1 测试方法描述

通过棋盘检测方法，检查每个 CPU 寄存器是否存在数值 0 或数值 1 的“停滞”故障。

a. 寄存器 R0~R12 有效位为 32 位，通过 0x5555 5555 和 0xAAAA AAAA 数据模式行比较；
b. 寄存器 R13 有效位为 30 位，通过 0x5555 5554 和 0xAAAA AAA8 数据模式行比较；
c. 寄存器 R14 有效位为 32 位，通过 0x5555 5555 和 0xAAAA AAAA 数据模式行比较；
d. Cortex-M0+的 APSR 寄存器有效位为位 28~31，通过 0x5000 0000 和 0xA00 0000 数据模式进行比较；
e. Cortex-M4 的 APSR 寄存器有效位为位 27~31，通过 0x5000 0000 和 0xA800 0000 数据模式进行比较；
f. 寄存器 CONTROL 有效位仅有位 1，通过 0x0000 0000 和 0x0000 0002 数据模式进行比较；
g. 寄存器 PRIMASK 有效位仅有位 0，通过 0x0000 0000 和 0x0000 0001 数据模式进行比较；
h. Cortex-M4 的 FAULTMASK 寄存器有效位仅有位 0，通过 0x0000 0000 和 0x0000 0001 数据模

式进行比较；

i. Cortex-M4 的 BASEPRI 寄存器有效位为位 7~4，通过 0x0000 0050 和 0x0000 00A0 数据模式进行比较。

测试流程如图 3-1 所示：

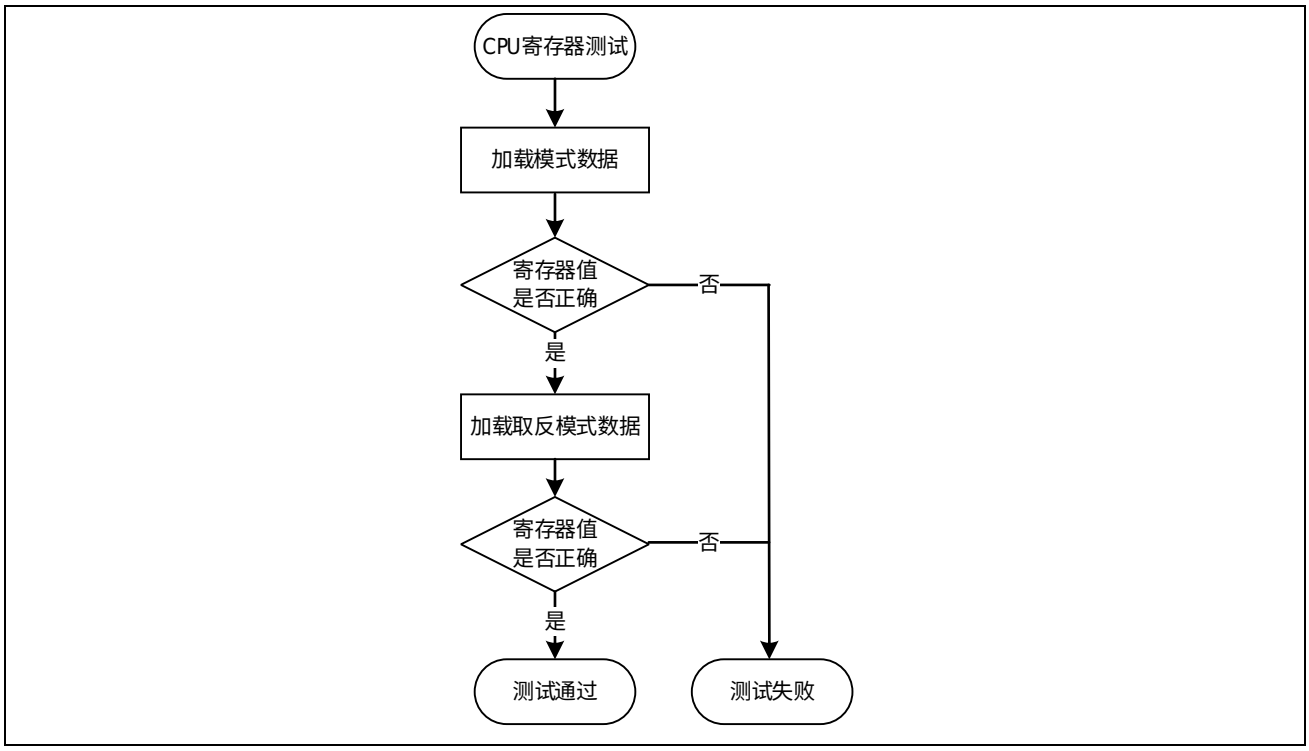

图 3-1 CPU 寄存器测试流程

3.1.2 库实现的函数接口

1. **[名称]**

STL_CpuTestStartup

[描述]

上电测试，包含寄存器 R0~R14、APSR、CONTROL、PRIMASK、FAULTMASK（Cortex-M4）、BASEPRI（Cortex-M4）。

注意：此函数运行会将 R4~R7 压入栈。

[参数说明]

输入：无。

输出：STL_OK：CPU 寄存器测试通过；

STL_ERR：CPU 寄存器测试失败。

[函数定义]

```
uint32_t STL_CpuTestStartup(void);
```

由于 HC32F420 的 BASEPRI 寄存器配置不同，不能直接调用库内的该函数测试。

2. **[名称]**

STL_CpuTestRuntime

[描述]

运行时测试，包含寄存器 R1~R12。

注意：此函数运行前会将 R4~R7 压入栈。

[参数说明]

输入：无。

输出：STL_OK：CPU 寄存器测试通过；

STL_ERR：CPU 寄存器测试失败。

[函数定义]

```
uint32_t STL_CpuTestRuntime(void);
```

3.2 PC 寄存器测试

Cortex-M0+和 Cortex-M4 CPU 的 PC 寄存器加载数据会导致程序跳转执行，所以不能通过棋盘检测方法测试。

3.2.1 测试方法描述

通过加载 PC 寄存器为不同区域子程序，跳转至子程序，并在子程序内保存子程序地址，然后返回主程序验证子程序地址是否正确。

测试流程如图 3-2 所示：

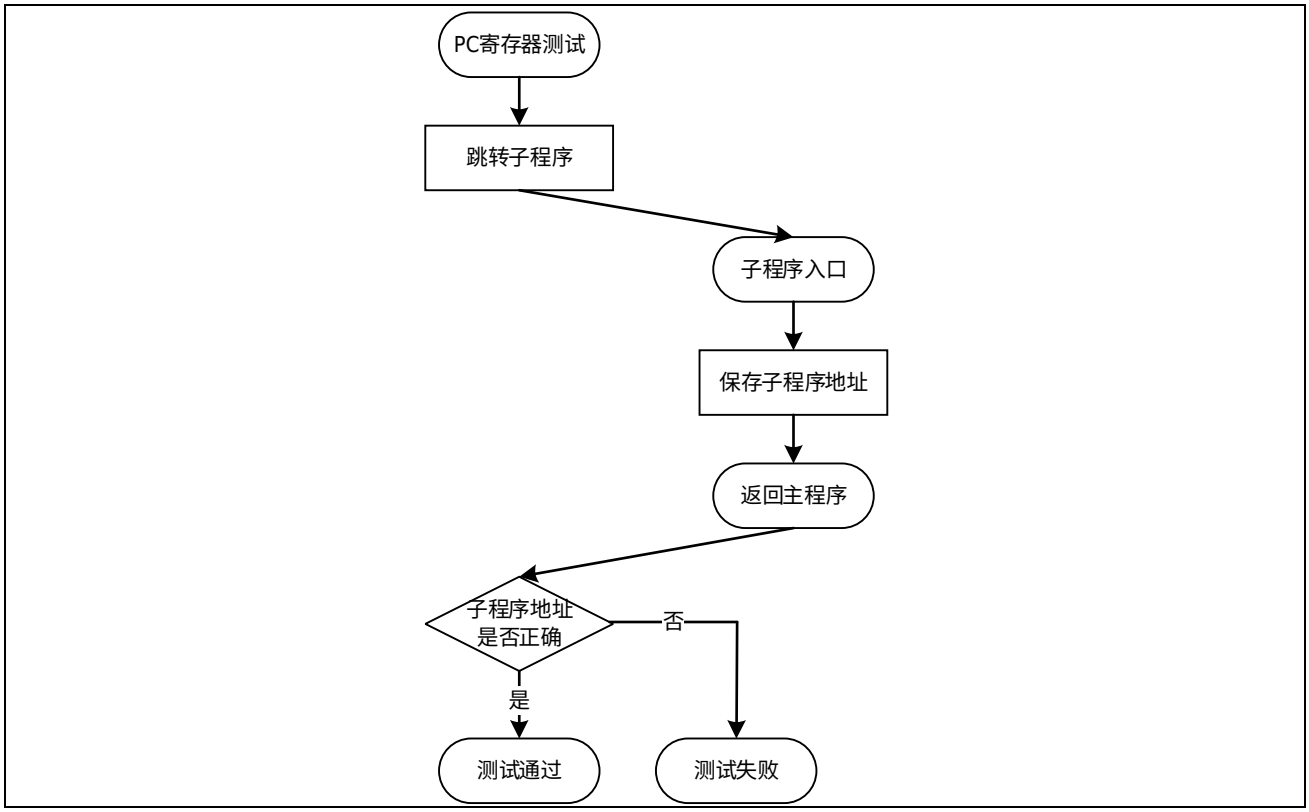

图 3-2 PC 寄存器测试流程

3.2.2 库实现的函数接口

1. **[名称]**

STL_PcTest

[描述]

上电和运行时测试。

[参数说明]

输入：无。

输出：STL_OK：CPU PC 寄存器测试通过；

STL_ERR：CPU PC 寄存器测试失败。

[函数定义]

```
uint32_t STL_PcTest(void);
```

3.3 ROM 测试

Flash 存储器故障/错误控制技术基于代表存储器中所有字内容的单字测试。

3.3.1 测试方法描述

通过相同算法 CRC32 计算校验和，并与保存的校验和比较。

测试流程如图 3-3 所示：

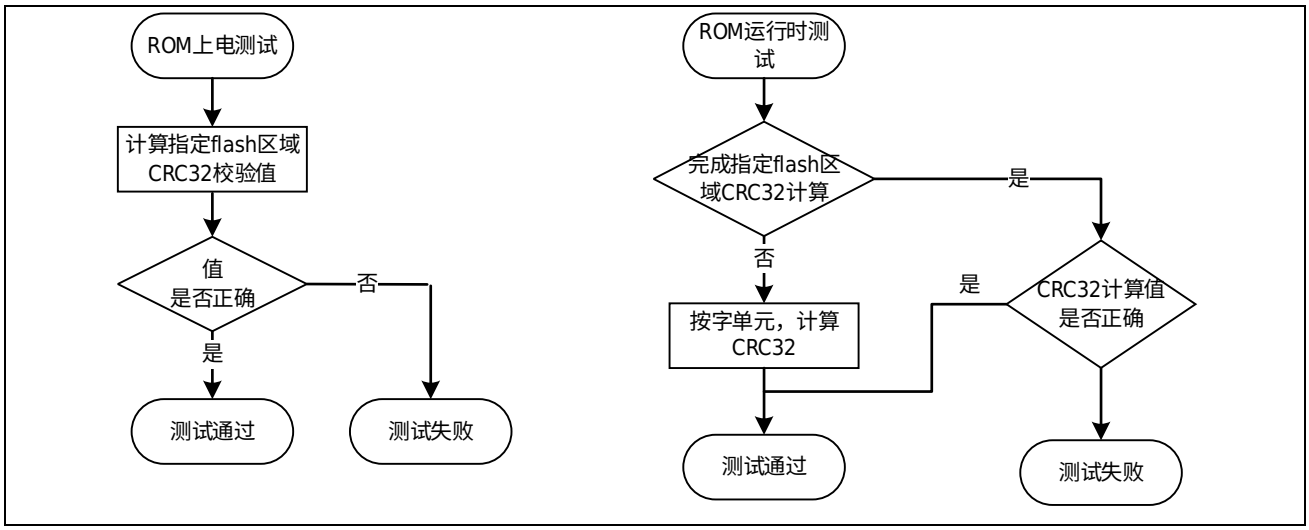

图 3-3 ROM 测试流程

3.3.2 库实现的函数接口

1. **[名称]**

STL_FlashStartupTest

[描述]

上电测试，验证指定 flash 区域 CRC32 校验值是否正确。

[参数说明]

输入：stc_flash_test_para_t *stcFlashTestPara。

输出：STL_OK：Flash 测试通过；

STL_ERR：Flash 测试失败。

[函数定义]

```
uint32_t STL_FlashStartupTest(stc_flash_test_para_t *stcFlashTestPara);
```

2. **[名称]**

STL_FlashRuntimeInit

[描述]

运行时测试，初始化 CRC32 计算值。

[参数说明]

输入：无。

输出：STL_OK：Flash 初始化通过；

STL_ERR：Flash 初始化失败。

[函数定义]

```
uint32_t STL_FlashRuntimeInit(void);
```

3. **[名称]**

STL_FlashRuntimeTest

[描述]

运行时测试，按字单元计算 CRC32 值，完成指定 flash 区域计算后，比较是否正确。

如有多个分区的 FLASH 需要测试，可以定义和配置多套输入传参结构体，并分别多次调用该测试函数。

[参数说明]

输入：stc_flash_test_para_t *stcFlashTestPara。

输出：STL_OK：Flash 测试通过；

STL_ERR：Flash 测试失败。

[函数定义]

```
uint32_t STL_FlashRuntimeTest(stc_flash_test_para_t *stcFlashTestPara);
```

4. **[名称]**

STL_Crc32Init

[描述]

Flash 测试 CRC32 校验值初始化。

[参数说明]

输入：无。

输出：无。

[函数定义]

```
void STL_Crc32Init(void);
```

5. **[名称]**

STL_GetCrc32

[描述]

返回 CRC32 最终校验值。

[参数说明]

输入：u32CalcCrc32Value：CRC32 校验值。

输出：CRC32 最终校验值。

[函数定义]

```
uint32_t STL_GetCrc32(uint32_t u32CalcCrc32Value);
```

6. **[名称]**

STL_StartupCalculateCRC32Value

[描述]

Flash 启动时测试 CRC32 校验函数。

[参数说明]

输入：u32Crc32Value：CRC32 校验初始值；

pu8Data：校验初始地址；

u32Len：校验区块长度。

输出：CRC32 校验值。

[函数定义]

```
uint32_t STL_StartupCalculateCRC32Value(uint32_t u32Crc32Value, uint8_t *pu8Data, uint32_t u32Len);
```

7. **[名称]**

STL_CalculateCRC32Value

[描述]

Flash 运行时测试 CRC32 校验函数。

[参数说明]

输入：u32Crc32Value：当次调用时 CRC32 校验初始值；

pu8Data：当次调用时校验初始地址；

u32Len：当次调用时校验区块长度。

输出：CRC32 校验值。

[函数定义]

```
uint32_t STL_CalculateCRC32Value(uint32_t u32Crc32Value, uint8_t *pu8Data, uint32_t u32Len);
```

3.3.3 用户实现的函数接口

1. **[名称]**

STL_FlashStartupParaInit

[描述]

上电测试时，对所需参数 stcFlashTestPara0 结构体进行初始化配置。

[参数说明]

输入：无。

输出：无。

配置参数：stc_flash_test_para_t　stcFlashTestPara0；

stcFlashTestPara0.u32TestFlashCrc32Start：Flash CRC32 校验的起始地址；

stcFlashTestPara0.u32TestFlashCrc32Size：Flash CRC32 校验的区域范围；

stcFlashTestPara0.u32TestFlashBuildCrc32：编译后得到并存储的 Flash CRC32 的校验值。

[函数定义]

```
void STL_FlashStartupParaInit(void);
```

2. **[名称]**

STL_FlashRuntimeParaInit

[描述]

运行时测试，对所需参数 stcFlashTestPara0 结构体进行初始化配置。

[参数说明]

输入：无。

输出：无。

结构体定义：

typedef struct

```
{
    uint32_t   u32TestFlashCrc32Start;
    uint32_t   u32TestFlashCrc32Size;
    uint32_t   u32TestFlashCrc32End;
    uint32_t   u32TestFlashBlockSize;
    uint32_t   u32TestFlashBuildCrc32;
    uint32_t   u32CheckAddr;
    uint32_t   u32CheckEndAddr;
    uint32_t   u32CalcLen;
    uint32_t   u32CalcCrc32Value;
}stc_flash_test_para_t;
```

配置参数：stc_flash_test_para_t stcFlashTestPara0;

stcFlashTestPara0.u32TestFlashCrc32Start：Flash CRC32 校验的起始地址；

stcFlashTestPara0.u32TestFlashCrc32End：Flash CRC32 校验的结束地址。起始地址和结束地址通常为整个程序代码区域，不包含运行时需要修改数据的 FLASH 区域；

stcFlashTestPara0.u32TestFlashCrc32Size：Flash CRC32 校验的区域范围，一般为结束地址减去起始地址；

stcFlashTestPara0.u32TestFlashBlockSize：主循环运行测试（RuntimeTest）时，单次调用 Flash 测试函数所校验的区域大小值；

stcFlashTestPara0.u32TestFlashBuildCrc32：整个待校验 Flash 区域，经编译后得到并存储在指定地址的 Flash CRC32 的校验值，用来与 RuntimeTest 时得到的 CRC32 校验结果做比对；

stcFlashTestPara0.u32CheckAddr：Flash CRC32 运行时当次校验的起始地址；

stcFlashTestPara0.u32CheckEndAddr：Flash CRC32 运行时当次校验的判断边界；

stcFlashTestPara0.u32CalcLen：Flash CRC32 运行时当次校验的长度；

stcFlashTestPara0.u32CalcCrc32Value：Flash CRC32 运行时当次校验的结果。

[函数定义]

```
void STL_FlashRuntimeParaInit(void);
```

3.4 RAM 测试

通过使用 March C 存储器测试检查 DC 故障。

3.4.1 测试方法描述

March C 测试方法如下：

- 步骤 1 – 将 RAM 写为全 0
- 步骤 2 – 从最低的地址开始，读取 0 且验证，写入 1，地址递增
- 步骤 3 – 从最低的地址开始，读取 1 且验证，写入 0，地址递增
- 步骤 4 – 从最高的地址开始，读取 0 且验证，写入 1，地址递减
- 步骤 5 – 从最高的地址开始，读取 1 且验证，写入 0，地址递减
- 步骤 6 – 从最低的地址开始，读取 0 且验证，地址递增

测试流程如图 3-4 所示：

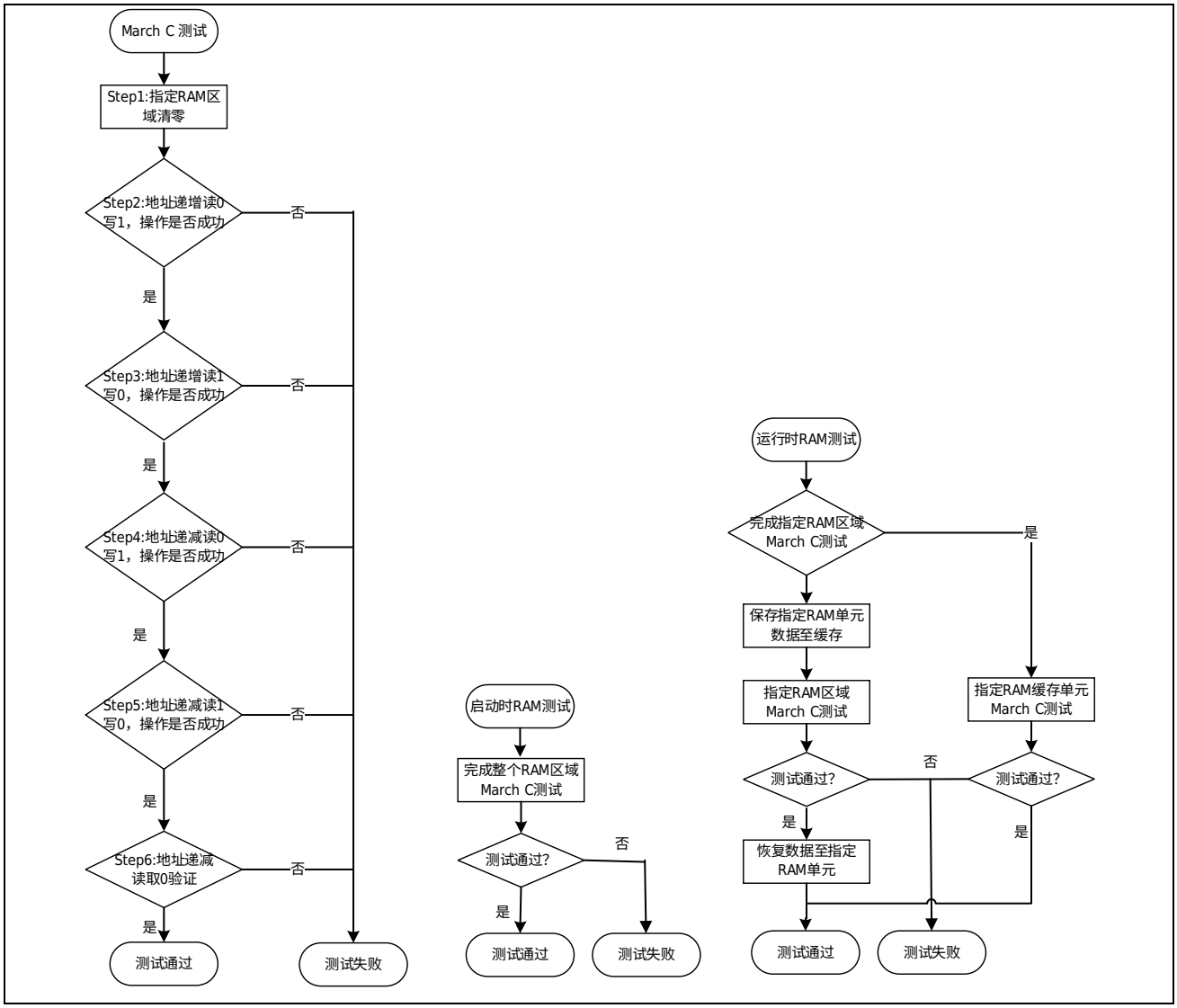

图 3-4 RAM 测试流程

3.4.2 库实现的函数接口

1. **[名称]**

STL_FullRamTestStartup

[描述]

上电测试，测试整个 RAM 区域。

注意：运行此测试函数，会将全片 RAM 清零，包含 STACK 区域，并且 R4~R6 寄存器的值会被改写！此函数运行不能被中断打断！

[参数说明]

输入：uint32_t u32StartAddr：RAM 起始地址；

uint32_t u32EndAddr：RAM 结束地址。

要求被测试的 RAM 是连续地址的区域。如有多块不连续地址的 RAM 区域，可以分多次调用该函数。

输出：STL_OK：RAM 测试通过；

STL_ERR：RAM 测试失败。

[函数定义]

```
uint32_t STL_FullRamTestStartup(uint32_t u32StartAddr, uint32_t u32EndAddr);
```

2. **[名称]**

STL_RamRuntimeInit

[描述]

运行时测试，指定 March C 测试 RAM 起始地址。

[参数说明]

输入：无。

输出：STL_OK：RAM 测试初始化通过。

[函数定义]

```
uint32_t STL_RamRuntimeInit(void);
```

3. **[名称]**

STL_RamRuntimeTest

[描述]

运行时测试，March C 测试指定 RAM 区域和缓存 Buff 备份区，指定 RAM 区域需要为连续地址的区域。

注意：此函数运行不能被中断打断！

[参数说明]

输入：无。

输出：STL_OK：RAM 测试通过；

STL_ERR：RAM 测试失败。

[函数定义]

```
uint32_t STL_RamRuntimeTest(void);
```

3.4.3 用户实现的函数接口

1. **[名称]**

STL_RamRuntimeParaInit

[描述]

运行时测试，指定 March C 测试 RAM 参数。

[参数说明]

输入：无。

输出：无。

结构体定义：

```
typedef struct
{
    uint32_t u32TestRamBufWords;
    uint32_t u32TestRamBckGrnd;
    uint32_t u32TestRamInvBckGrnd;
    uint32_t u32TestRamMarchRamStart;
    uint32_t u32TestRamMarchRamEnd;
}stc_ram_test_para_t;
```

配置参数：stc_ram_test_para_t　　stcRamTestPara;

stcRamTestPara.u32TestRamMarchRamStart：设置 RAM 测试区域的起始地址；

stcRamTestPara.u32TestRamMarchRamEnd：设置 RAM 测试区域的结束地址；

stcRamTestPara.u32TestRamBckGrnd：MARCH C RAM 测试数据；

stcRamTestPara. u32TestRamInvBckGrnd：MARCH C RAM 测试数据；

stcRamTestPara.u32TestRamBufWords：设置 RAM 测试缓存 buff word 长度。

[函数定义]

```
void STL_RamRuntimeParaInit(void);
```

3.5 Stack 测试

3.5.1 测试方法描述

链接器指定 Stack 边界域，通过该域值判断是否堆栈溢出

测试流程如图 3-5 所示：

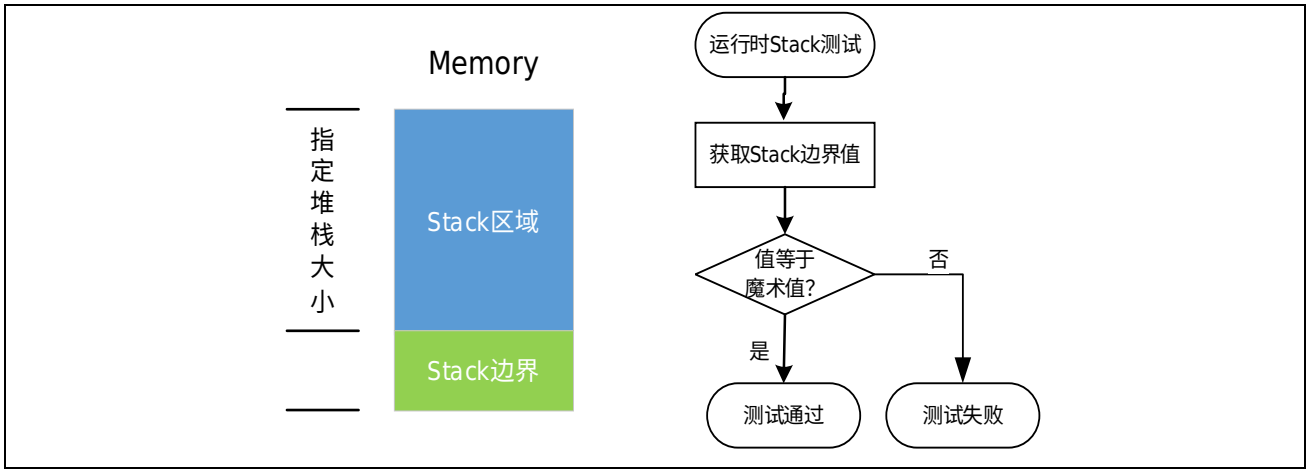

图 3-5 Stack 测试流程

3.5.2 库实现的函数接口

1. **[名称]**

STL_StackRuntimeInit

[描述]

设置 Stack 边界魔术值。

[参数说明]

输入：无。

输出：STL_OK：STACK 测试初始化成功。

[函数定义]

```
uint32_t STL_StackRuntimeInit(void);
```

2. **[名称]**

STL_StackRuntimeTest

[描述]

运行时测试，检测 Stack 边界魔术值是否正确。

[参数说明]

输入：无。

输出：STL_OK：Stack 正常；

STL_ERR：Stack 溢出。

[函数定义]

```
uint32_t STL_StackRuntimeTest(void);
```

3.6 时钟和中断测试

3.6.1 测试方法描述

时钟和中断测试：

使用两个独立时钟：系统时钟和低速时钟。通过一个使用系统时钟的定时器和一个使用低速时钟的定时器，在系统时钟的定时器中断内计数，在低速时钟定时器的中断内对该计数进行上限和下限的判断，监测系统时钟是否正常，也监测中断数量是否在设定范围内。

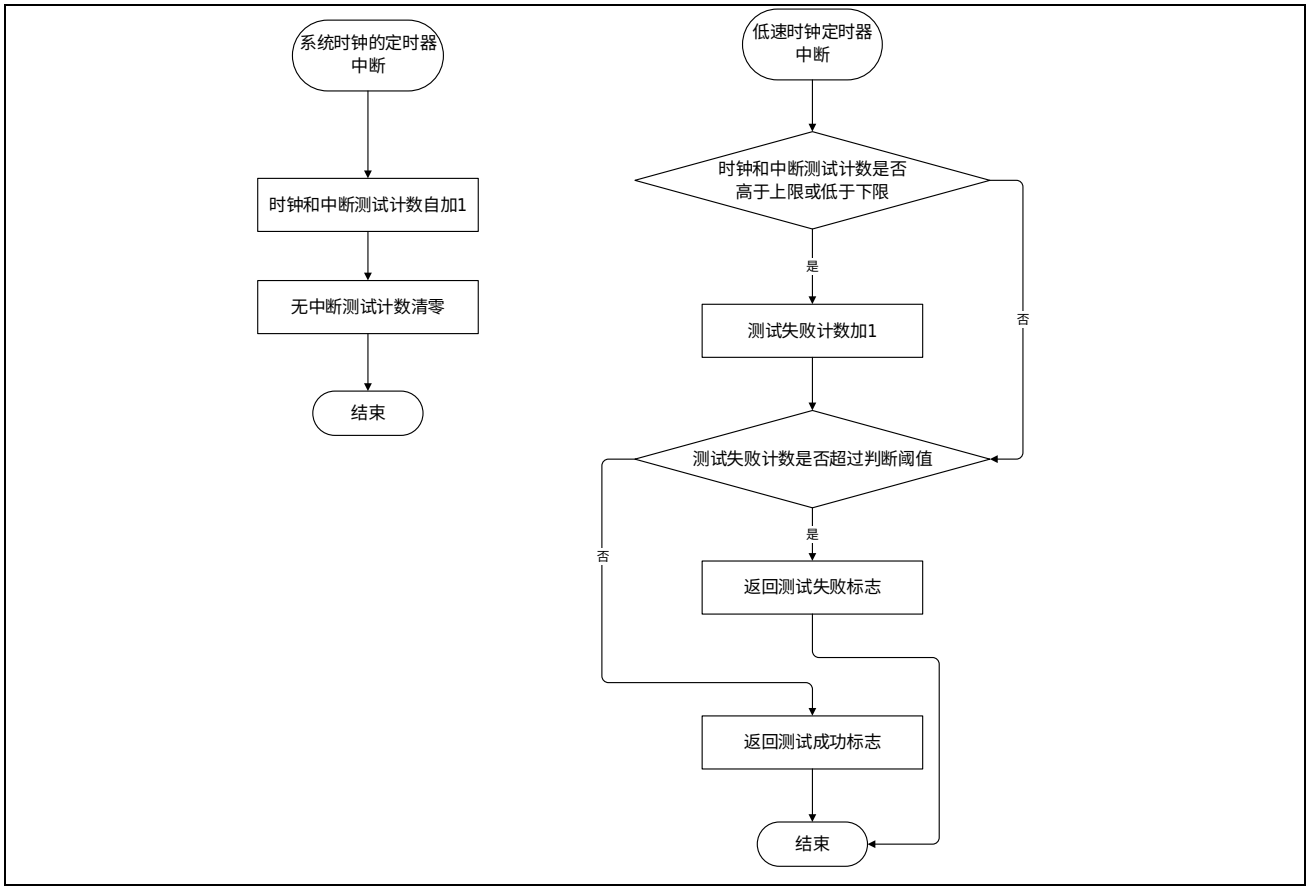

图 3-6　时钟和中断测试流程

无中断测试：

在 main 主循环内对无中断测试计数变量进行自加，在系统时钟的定时器中断内对该变量进行清零，然后在 main 主循环的测试函数内对该变量进行判断，如果没有中断产生，该变量超过设定的上限值就返回测试错误，否则返回测试正常。

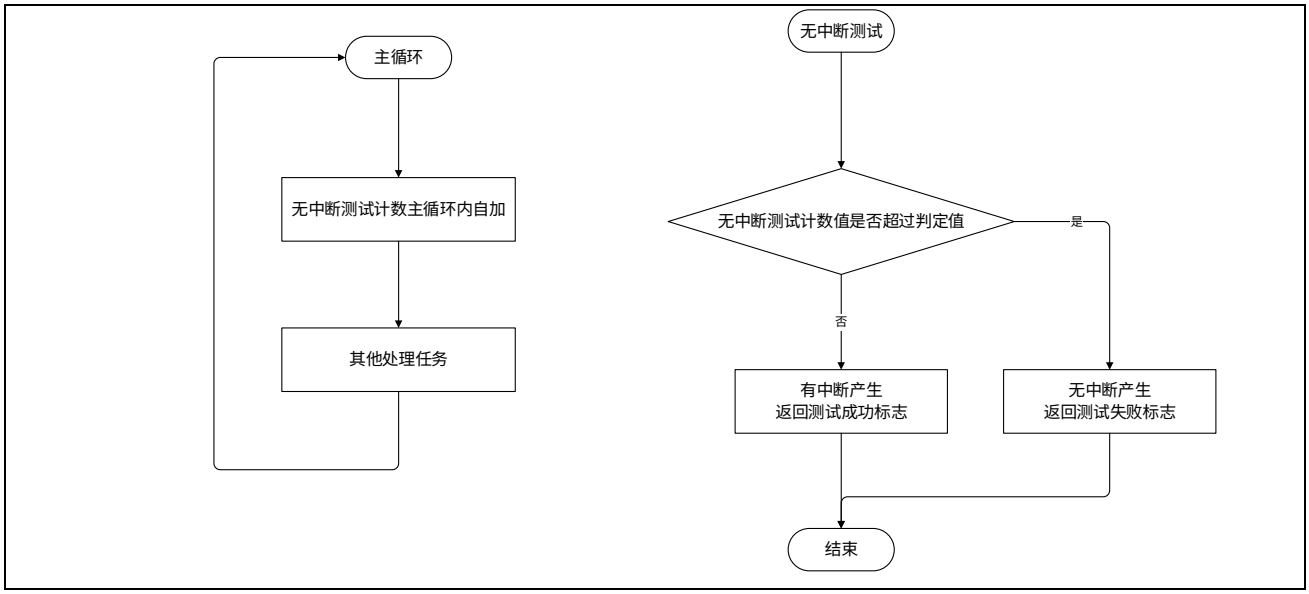

图 3-7　无中断测试流程

3.6.2 库实现的函数接口

1. **[名称]**

STL_SysTimerIrq

[描述]

系统时钟的定时器中断内对时钟和中断计数值计数，并且对无中断测试计数值清零。

[参数说明]

输入：无。

输出：无。

[函数定义]

```
void STL_SysTimerIrq(void);
```

2. **[名称]**

STL_ClockTest

[描述]

在低速时钟定时器中断内，对时钟和中断测试计数值进行判断，如果计数值在上下限内，则返回测试成功，如果不在上下限内的次数超过判断阈值，则返回测试失败。

[参数说明]

输入：无。

输出：STL_OK：测试正常；

STL_ERR：测试失败。

[函数定义]

```
uint32_t STL_ClockTest(void);
```

3. **[名称]**

STL_ClockStartupTest

[描述]

启动测试时调用，初始化测试，并等待测试结束，若测试失败标志为 1，则返回测试失败，否则返回测试正常。并将时钟和中断测试去初始化。

[参数说明]

输入：无。

输出：STL_OK：测试正常；

STL_ERR：测试失败。

[函数定义]

```
uint32_t STL_ClockStartupTest(void);
```

4. **[名称]**

STL_ClockRuntimeTest

[描述]

运行测试时调用，判断测试失败标志，若为 1，则返回测试失败，否则返回测试正常。

[参数说明]

输入：无。

输出：STL_OK：测试正常；

STL_ERR：测试失败。

[函数定义]

```
uint32_t STL_ClockRuntimeTest(void);
```

5. **[名称]**

STL_NonIntTestRuntime

[描述]

运行测试时调用，判断无中断测试计数值是否超过预设值。如果超过该预设值，表示长时间没有系统定时器中断请求并进入中断处理函数，则返回测试失败，否则返回测试正常。

[参数说明]

输入：无。

输出：STL_OK：测试正常；

STL_ERR：测试失败。

[函数定义]

```
uint32_t STL_NonIntTestRuntime(void);
```

3.6.3 用户实现的函数接口

1. **[名称]**

STL_ClockParaInit

[描述]

时钟和中断测试参数初始化，需要用户根据实际工程调整测试参数。

[参数说明]

输入：无。

输出：无。

结构体定义：

```
typedef struct
{
    uint16_t   u16ClkTestUpperLimit;
    uint16_t   u16ClkTestLowerLimit;
    uint16_t   u16ClkTestIrqCnt;
    uint16_t   u16TestFailCnt;
    uint32_t   m_u32TmrCount;
    uint32_t   u32NoneIntTestCnt;
    uint32_t   u32NonIntTestJudge;
    uint8_t    u8TestFailJudge;
    uint8_t    u8FlagClockTestFail;
}stc_clock_test_para_t;
```

配置参数：stc_clock_test_para_t stcClockTestPara;

stcClockTestPara.u16ClkTestUpperLimit：设置时钟和中断测试计数判断上限值，一个低速定时器定时周期内，允许的高速定时器定时周期的次数上限，样例为正常周期次数的 1.2 倍；

stcClockTestPara.u16ClkTestLowerLimit：设置时钟和中断测试计数判断下限值，一个低速定时器定时周期内，允许的系统时钟定时器定时周期的次数下限，样例为正常周期次数的 0.8 倍；

stcClockTestPara.m_u32TmrCount：时钟和中断测试计数值，用于系统时钟定时器内计数，初始化时清零；

stcClockTestPara.u16TestFailCnt：时钟和中断测试失败次数，当 stcClockTestPara.u32TmrCount 超过所设置的上限或低于所设置的下限，即失败次数累积 1 次，初始化时清零；

stcClockTestPara.u8TestFailJudge：设置时钟和中断测试失败次数判断阈值，在低速时钟定时器内调用 STL_ClockTest（）函数进行判断，当 stcClockTestPara.u16TestFailCnt 超过该判断阈值时，即表示测试失败；

stcClockTestPara.u8FlagClockTestFail：时钟和中断测试失败标志，用于标记，当测试失败时，该 flag 会置 1，初始化时清零；

stcClockTestPara.u16ClkTestIrqCnt：时钟和中断启动测试的重复测试次数，设置范围为 0~3。例如：初始化为 0 的时候，需要等待重复 4 次测试，初始化为 2 的时候，需要等待重复 2 次测试；

stcClockTestPara.u32NoneIntTestCnt：无中断测试计数，在主循环内计数，在系统时钟定时器中断处理函数内清零，初始化时清零；

stcClockTestPara.u32NonIntTestJudge：设置无中断测试计数判断阈值，在 RuntimeTest 测试内判断 stcClockTestPara.u32NoneIntTestCnt 的值是否超过该阈值，如超过该阈值则返回测试失败。

[函数定义]

```
void STL_ClockParaInit(void);
```

2. **[名称]**

STL_ClockTestInit

[描述]

调用参数初始化，并且配置系统时钟的定时器和低速时钟的定时器。

[参数说明]

输入：无。

输出：STL_OK：配置正常；

STL_ERR：配置失败。

[函数定义]

```
uint32_t STL_ClockTestInit(void);
```

3. **[名称]**

SysClktimer_IRQHandler

[描述]

系统时钟的定时器中断，调用 STL_SysTimerIrq 函数。

[参数说明]

输入：无。

输出：无。

[函数定义]

```
void SysClktimer_IRQHandler(void);
```

4. **[名称]**

LrcClktimer_IRQHandler

[描述]

低速时钟的定时器中断，调用 STL_ClockTest 函数，若该函数返回值为 STL_ERR，则将时钟和中断测试失败标志置为 1。

[参数说明]

输入：无。

输出：无。

[函数定义]

```
void LrcClktimer_IRQHandler(void);
```

3.7 模数转换测试

模数转换测试通过扫描模式，AD 转换比较，验证 AD 转换准确性。

3.7.1 测试方法描述

模数转换测试通过选择一个基准电压，采样电源电压的分压；或者使用电源电压为参考电源，采样一路基准电压，比较 AD 转换后数值是否在预期范围内。

模数转换测试流程如图 3-8 所示：

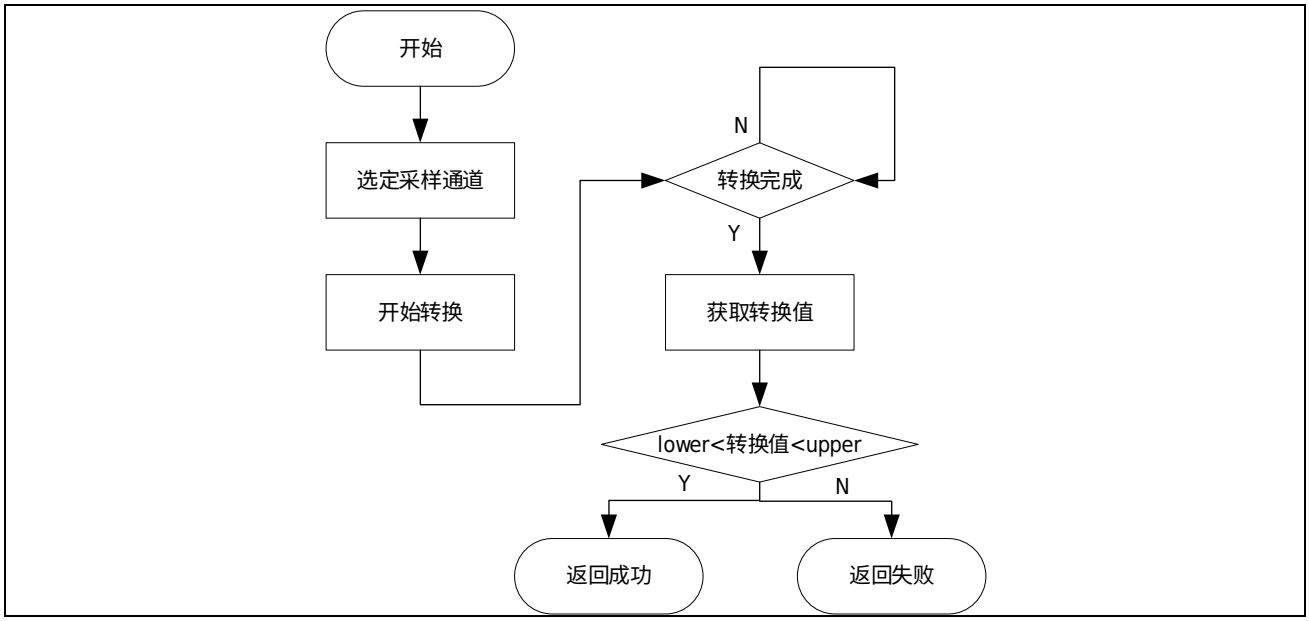

图 3-8 ADC 测试流程

3.7.2 库实现的函数接口

1. **[名称]**

STL_AdcTestStartup

[描述]

ADC 测试，将测试结果读取并取平均值，然后与预设的上下限值作比较，如果在上下限之内，则返回测试正常，否则返回测试失败。

[参数说明]

输入：无。

输出：STL_OK：测试正常；

STL_ERR：测试失败。

[函数定义]

```
uint32_t STL_AdcTestStartup(void);
```

3.7.3 用户实现的函数接口

1. **[名称]**

STL_AdcTestParaInit

[描述]

ADC 测试参数初始化。

[参数说明]

输入：无。

输出：无。

结构体定义：

```
typedef struct
{
    uint32_t u32AdcFlagReg;
    uint32_t u32AdcFlagMsk;
    uint32_t u32AdcFlagClrReg;
    uint32_t u32AdcFlagClrMsk;
    uint32_t u32AdcResult0Reg;
    uint32_t u32AdcResult1Reg;
    uint32_t u32AdcResult2Reg;
    uint32_t u32AdcResult3Reg;
    uint32_t u32AdcTestUpperLimit;
    uint32_t u32AdcTestLowerLimit;
} stc_stl_adc_test_para_t;
```

配置参数：stc_stl_adc_test_para_t stcAdcTestPara;

stcAdcTestPara. u32AdcFlagReg：将 ADC 中断状态标志寄存器地址[注1] 赋值给该成员；

stcAdcTestPara.u32AdcFlagMsk：ADC 中断状态标志 MASK，设置需要查询的中断状态标志位在中断状态寄存器的 MASK 值。例如需要查询的中断标志在 bit4，那么 MASK 值为 0x0000 0010UL；

stcAdcTestPara.u32AdcFlagClrReg：将 ADC 中断状态标志清零寄存器地址[注1] 赋值给该成员；

stcAdcTestPara.u32AdcFlagClrMsk：ADC 中断状态标志清零 MASK，设置需要清零的中断状态标志位在中断状态标志清零寄存器的 MASK 值。例如需要清零的中断标志在 bit4，那么 MASK 值为 0x0000 0010UL；

stcAdcTestPara.u32AdcResult0Reg：通常将 ADC 转换结果（数据）寄存器 0 的地址赋值给该成员。如果某型号 ADC 没有转换结果寄存器组，只有一个转换结果寄存器，那么可以定义一个 32bit 的数组，将数组成员 0~3 地址分别赋值给 stcAdcTestPara.u32AdcResult0Reg~stcAdcTestPara.u32AdcResult3Reg，通过 DMA 将 ADC 转换完的结果传输到该数组；

stcAdcTestPara.u32AdcResult1Reg：ADC 转换结果（数据）寄存器 1；

stcAdcTestPara.u32AdcResult2Reg：ADC 转换结果（数据）寄存器 2；

stcAdcTestPara.u32AdcResult3Reg：ADC 转换结果（数据）寄存器 3；

stcAdcTestPara.u32AdcTestUpperLimit：设置 ADC 转换结果上限，如果最终转换结果平均值超过该上限，则返回测试失败；

stcAdcTestPara.u32AdcTestLowerLimit：ADC 转换结果下限，如果最终转换结果平均值低于该下限，则返回测试失败。

注 *1*：寄存器地址需要 *word* 对齐（末尾数字应是 *0*、*4*、*8*、*C*），如果某 *MCU* 的对应寄存器地址不是 *word* 对齐的，需要向前补齐地址：例如某款 *MCU* 对应的寄存器地址为 *0x4000 B846*，*MASK* 值为 *0x0001*，那么此处的地址需设置 *0x4000 B844*，相应的 *MASK* 值也做偏移处理为 *0x0001 0000*。

[函数定义]

```
static void STL_AdcTestParaInit(void);
```

2. **[名称]**

STL_AdcTestStartupInit /STL_AdcstartupInit

[描述]

ADC 测试配置并且启动 ADC 转换。

[参数说明]

输入：无。

输出：STL_OK：配置正常；

STL_ERR：配置失败。

[函数定义]

```
void STL_AdcTestStartupInit(void);
```

3.8 看门狗测试

看门狗测试主要验证看门狗复位功能。

3.8.1 测试方法描述

看门狗测试启动看门狗复位，查看复位源是否由看门狗产生。

看门狗测试流程如图 3-9 所示：

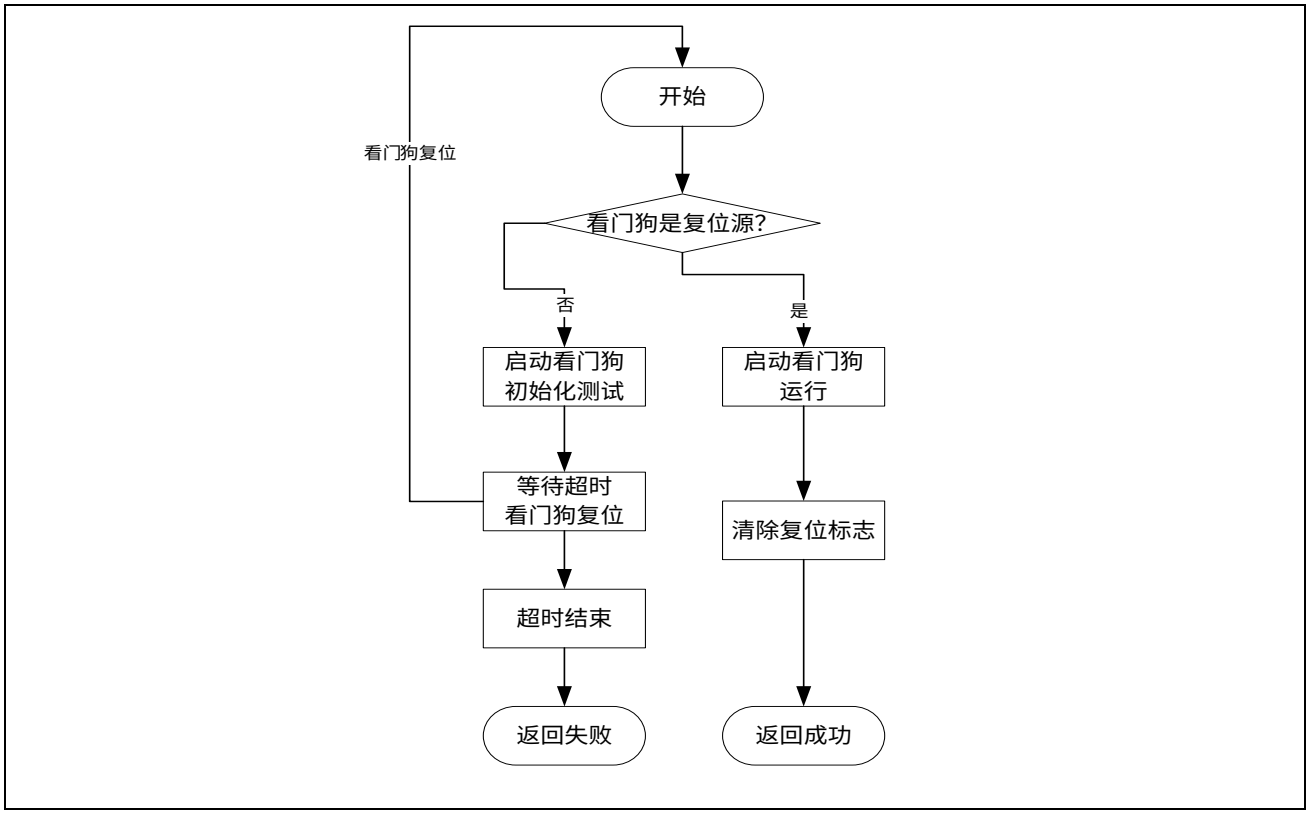

图 3-9 WDT 测试流程

3.8.2 库实现的函数接口

1. **[名称]**

STL_WdtStartupTest

[描述]

看门狗启动测试。上电或者复位后第一次进入测试函数，调用启动测试函数，并且等待看门狗复位，如超过预设的超时时间没有发生看门狗复位，则返回测试失败。当看门狗复位后第二次进入函数，查看是否有看门狗复位标志，如果有该标志置起则返回测试成功，否则再次调用启动测试函数。

[参数说明]

输入：无。

输出：STL_OK：看门狗复位功能正常；

STL_ERR：看门狗复位功能异常。

[函数定义]

```
uint32_t STL_WdtStartupTest(void);
```

3.8.3 用户实现的函数接口

1. **[名称]**

STL_WdtStartupParaInit

[描述]

看门狗测试参数初始化。

[参数说明]

输入：无。

输出：STL_OK：运行正常。

结构体定义：

```
typedef struct
{
    uint32_t u32RstFlagRge;
    uint32_t u32RstFlagRgeWdtMsk;
    uint32_t u32RstFlagRgeWdtClr;
    uint32_t u32WdtTimeout;
}stc_wdt_test_para_t;
```

配置参数：stc_wdt_test_para_t stcWdtTestPara;

stcWdtTestPara.u32RstFlagRge：将复位标志寄存器地址[注1]赋值给该成员；

stcWdtTestPara.u32RstFlagRgeWdtMsk：WDT 复位标志 MASK，如 WDT 复位标志在复位标志寄存器的 bit3，那么 MASK 值为 0x0000 0008UL；

stcWdtTestPara.u32RstFlagRgeWdtClr：WDT 复位标志清零 MASK，如 WDT 复位标志清零复位标志寄存器在 bit3，那么 MASK 值为 0x0000 0008UL；

stcWdtTestPara.u32WdtTimeout：WDT 测试超时时间（ms），该时间需大于在启动测试时的 WDT 复位时间。

***注 1*：寄存器地址需要 *word* 对齐（末尾数字应是 *0*、*4*、*8*、*C*），如果某 *MCU* 的对应寄存器地址不是 *word* 对齐的，需要向前补齐地址：例如某款 *MCU* 对应的寄存器地址为 *0x4000 B846*，*MASK* 值为 *0x01*，那么此处的地址需设置 *0x4000 B844*，相应的 *MASK* 值也做偏移处理为 *0x0001 0000*。**

[函数定义]

```
uint32_t STL_WdtStartupParaInit(void);
```

2. **[名称]**

STL_WdtStartupTestInit

[描述]

看门狗启动测试初始化配置。

[参数说明]

输入：无。

输出：STL_OK：运行正常。

[函数定义]

```
uint32_t STL_WdtStartupTestInit(void);
```

3. **[名称]**

STL_WdtRuntimeInit

[描述]

看门狗启动测试成功之后的运行配置。

[参数说明]

输入：无。

输出：STL_OK：运行正常。

[函数定义]

```
uint32_t STL_WdtRuntimeInit(void);
```

4. **[名称]**

STL_WdtTestStart

[描述]

看门狗启动。

[参数说明]

输入：无。

输出：STL_OK：运行正常。

[函数定义]

```
uint32_t STL_WdtTestStart(void);
```

5. **[名称]**

STL_WdtRuntimeFeed

[描述]

看门狗喂狗。

[参数说明]

输入：无。

输出：STL_OK：运行正常。

[函数定义]

```
uint32_t STL_WdtRuntimeFeed(void);
```

4 STL 样例工程

STL 根据 IAR 和 Keil IDE 提供两个演示项目。

4.1 代码介绍

STL 库测试分为两个主要过程：启动和运行时的测试。样例工程测试流程如图 4-1 所示。

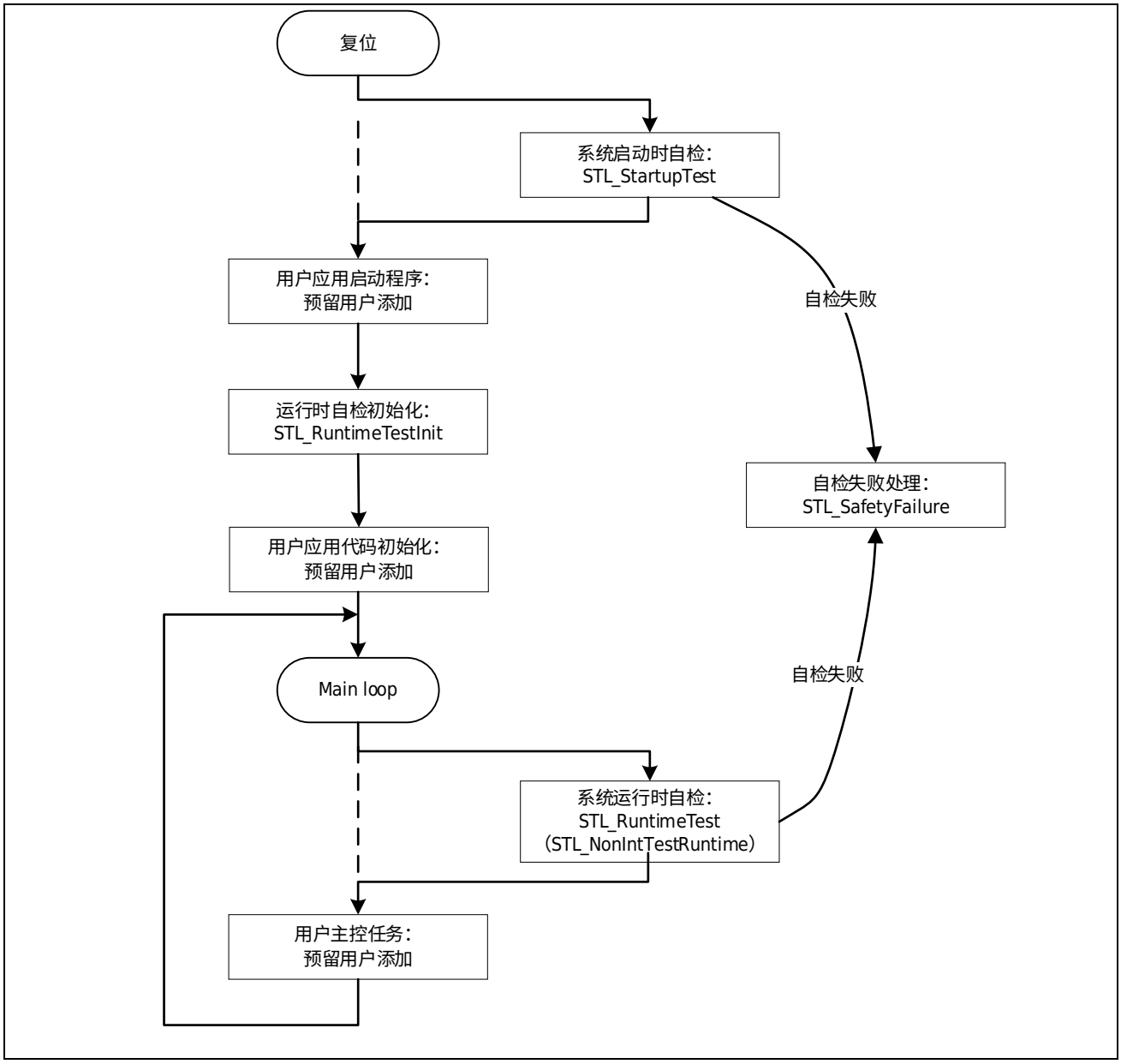

图 4-1 STL 样例流程

4.2 代码运行

用户可以通过小华半导体官网或客户支持途径得到到 STL 的样例代码。

以下部分主要介绍如何通过 IAR EWARM 编译、运行样例代码并观察结果：

- 确认安装正确的 IAR EWARM v7.7 工具（请从 IAR 官方网站下载相应的安装包，并参考用户手册进行安装）。
- 获取对应 MCU 型号的开发板。
- 从小华半导体官网或客户支持途径获取 STL 代码。
- 下载并运行项目文件：

  1) 打开项目工程，并打开‘main.c’如图 4-2 所示：

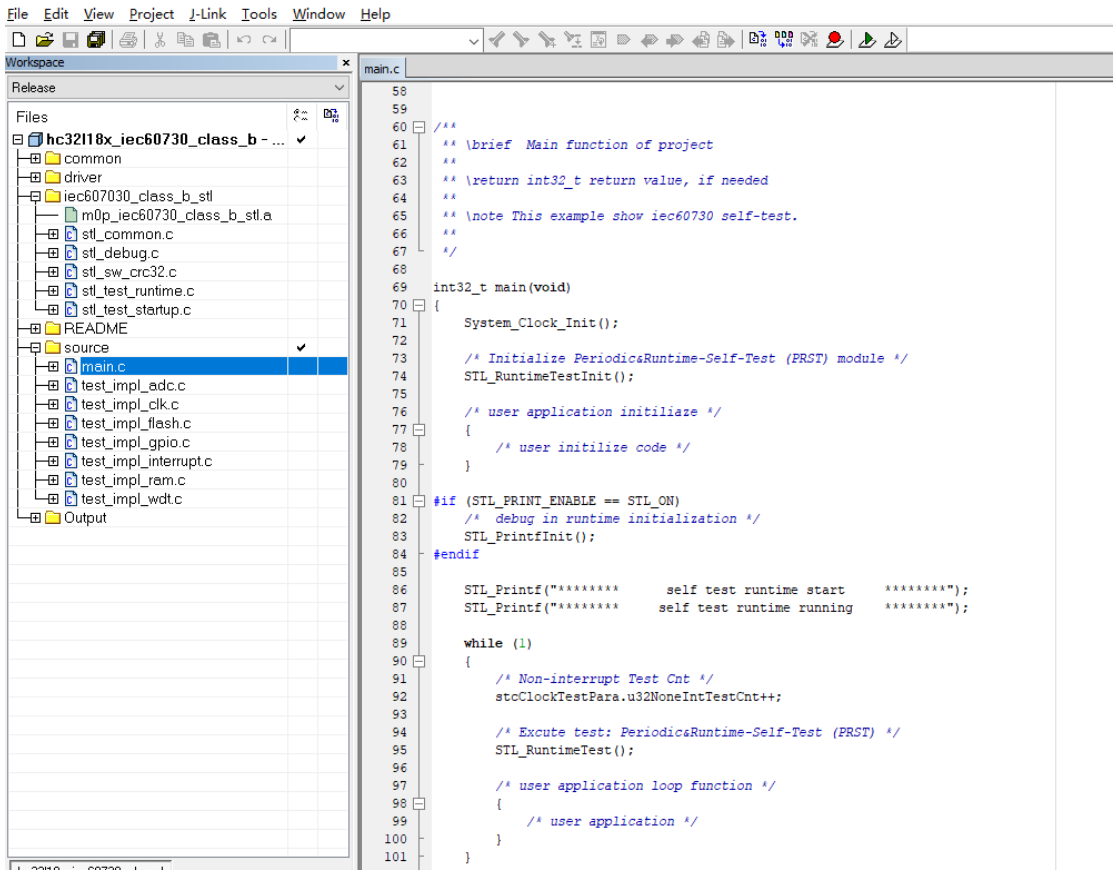

图 4-2 STL IAR 工程

  2) 点击  重新编译整个项目；

  3) 通过串口助手连接开发板虚拟端口；

  4) 点击  将代码下载到评估板上，全速运行；

  5) 观察串口助手打印信息如图 4-3 所示和测试板 LED 状态，若 LED 闪烁，表示测试失败。

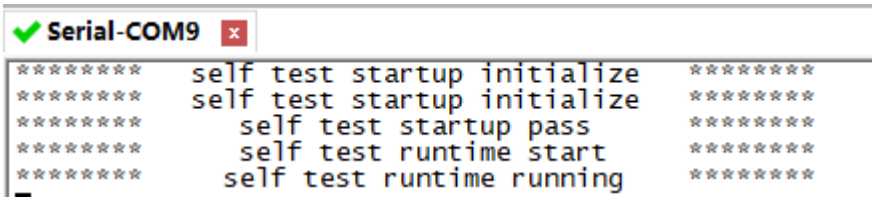

图 4-3 测试提示信息

版本修订记录

版本号	修订日期	修订内容
Rev1.00	2024/12/05	初版发布。
Rev1.01	2025/04/02	适用对象章节，新增 HC32A448、HC32A472、HC32A460、HC32A4A0 系列型号。
Rev1.02	2025/04/11	适用对象添加 HC32M441 系列。
Rev1.10	2026/03/24	手册“AN_基于 HC32_M4 内核的 IEC60730_ClassB_STL_Rev1.02”与“AN_基于 HC32_M0P 内核的 IEC60730_ClassB_STL_Rev1.00”合并，并补充适应 STL 库更新内容，合并后手册升级变更为：“AN_IEC60730_Class_B_STL 应用笔记_Rev1.10”。
Rev1.11	2026/08/04	适用对象新增 HC32M413、HC32F431、HC32M458 系列型号。