HT32 UL/IEC 60730-1 Class B 安全测试库使用指南

文件编码：AN0554SC

简介

国际电工委员会（IEC）制定了家电开发的安全标准 IEC 60730。IEC 60730-1 标准（家用和类似用途的自动电气控制器－第 1 部分：通用要求）规定了测试和诊断方法，以确保家电中受控设备的安全操作。附录 H 是该标准的关键部分，将软件分为 Class A、Class B、Class C 三类。Holtek 为用于防止设备发生不安全操作的 Class B 控制功能（例如洗衣设备的热熔断装置和门锁）提供了 HT32 安全测试库（Safety Test Library，以下简称 HT32 STL）。Class B 适用于大部分家电产品，包括洗碗机、洗衣机、电冰箱、冰柜和炊具等。根据 IEC 60730 标准，家电制造商必须按照 Class B 规则设计他们的产品。

本文描述了用于 IEC 60730-1 Class B 认证的 Holtek HT32 STL 的功能、环境、系统架构和使用方法。HT32 STL 适用于几乎所有的 HT32 Cortex®-Mx 系列 32-bit MCU，提供 Class B 自检功能，包括 CPU 寄存器、程序计数器、中断、时钟、非易失性存储器（Flash 存储器）、易失性存储器（SRAM）、数字 I/O、模拟 I/O 等测试项目，这些在 IEC60730-1 附录 H 的表 H11.12.7 中有定义。HT32 STL 可以帮助用户减少自检功能的开发时间，加速认证过程。

功能说明

UL/IEC 60730-1 Class B 自检功能

IEC 60730-1 附录 H 中的表 H11.12.7 定义了下列需要测试的 Class B 元件。HT32 STL 中都有包括。

元件	测试目的和 HT32 STL 使用方法
1.1 CPU 寄存器	使用固件定期自检，确认所有 CPU 寄存器正常工作，没有“卡住”。
1.3 程序计数器	使用固件定期自检，确认程序计数器正常工作，没有“卡住”。
2.0 中断	使用时隙监控，检测“无中断或过于频繁中断”的情况。
3.0 时钟	使用时隙监控，检测“错误频率”。
4.1 非易失性存储器	使用校验和，检测“所有单位（single bit）故障”。
4.2 易失性存储器	使用定期静态存储器测试（March C/X），检测“直流故障”。
7.1 数字 I/O	使用合理性检查，检测指定的故障状态。
7.2 模拟 I/O	使用合理性检查，检测指定的故障状态。

HT32 STL 特性

HT32 STL 包括以下特性。

1. 分层结构
2. 模块化和可配置
3. 易于使用和集成
4. 支持几乎所有的 HT32 系列单片机（Cortex®-Mx 内核）
   注：HT32 STL 需要低速定时/计数器进行时钟测试。因此不支持没有 RTC 或低速定时器的单片机（例如 HT32F52220/52230）。如需进一步了解，请咨询当地代理机构。

系统需求

MCU 硬件	HT32 STL 与下列 MCU 硬件有关。 **CPU 内核** ✓ 32 位 Arm® Cortex®-Mx 处理器内核 ✓ 内建嵌套向量中断控制器（NVIC） **片上存储器** ✓ 片上 Flash 存储器用于指令/数据和选项的存储 ✓ 片上 SRAM 用于变量的存储 **时钟控制单元 – CKCU** ✓ 外部高速晶体振荡器 ✓ 外部低速晶体振荡器 ✓ 内部高速 RC 振荡器 ✓ 内部低速 RC 振荡器 **模拟/数字转换器 – ADC** ✓ SAR ADC 内核 ✓ 外部和内部模拟输入通道 **I/O 端口 – GPIO** ✓ 多个 GPIO 引脚 ✓ Port X 映射为 16 个外部中断（EXTI） ✓ 支持输入使能功能用于输出引脚的回送检查 **实时时钟 – RTC** ✓ 带可编程预分频器的 32-bit 向上计数器 ✓ 中断和唤醒事件 **循环冗余校验 – CRC（可选）** ✓ 支持 CRC16、CCITT CRC16 和 IEEE-802.3 CRC32 多项式
编译环境	MDK-ARM V5 及以上版本 ARMCC V5（包含在 MDK-ARM IDE 中）
其它工具	✓ Windows 命令解释器 (cmd.exe) ✓ Gsar V1.21 (http://gnuwin32.sourceforge.net/packages/gsar.htm) ✓ SRecord V1.64 (http://srecord.sourceforge.net/) 注：Gsar 和 SRecord 包含在 HT32 固件库中，HT32 UL/IEC 60730-1 Class B STL 是以原始 Windows 二进制版本（.exe 文件）发布的。Holtek 没有修改任何源代码。详情请参考相应的版权信息。

环境设置

本节介绍如何设置 HT32 STL 的测试环境，包括硬件、软件和基本测试。

硬件

准备一个 Starter Kit 和一根 USB 数据线来下载和测试 HT32 STL 库。以 HT32F50241 Starter Kit 为例，板上有两个 USB 接口。使用 USB 数据线将计算机连接到板上 e-Link32 Pro 接口，如下图红色方框所示。Starter Kit 由 USB 接口供电。

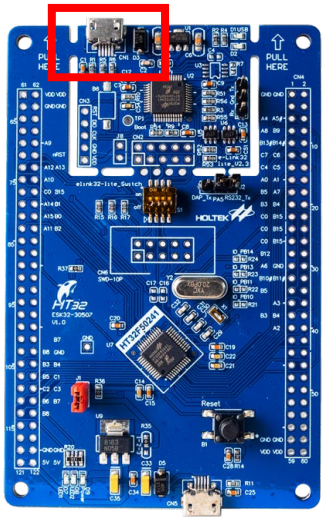


图 **1. ESK32-30507 - HT32F50241 Starter Kit**

软件

在使用 HT32 STL 之前，需要从 Holtek 网站下载最新的 Holtek HT32 固件库。下载位置如图 2 所示。下载后解压文件。

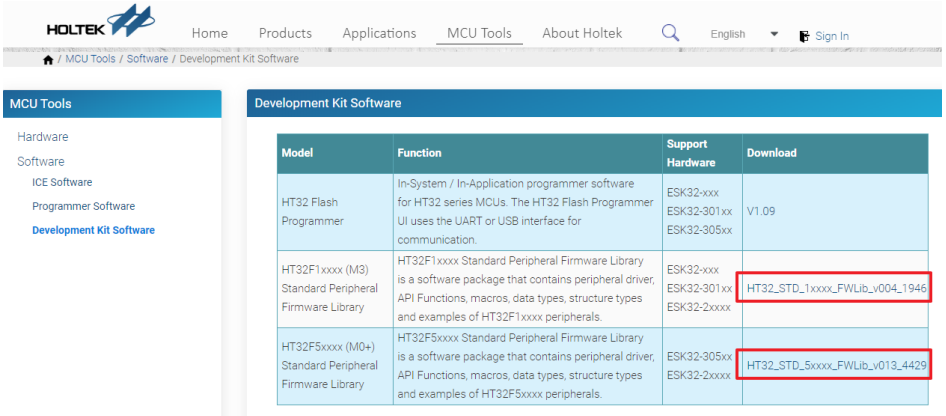


图 2. HT32 固件库下载链接

注意，HT32 STL 要求 HT32 固件库的最低版本如下：

- ✓ Cortex®-M0+：HT32_STD_5xxxx_FWLib_**v012_4285**
- ✓ Cortex®-M3：HT32_STD_1xxxx_FWLib_**v004_1946**

通过下面的链接下载 HT32 STL 应用代码。HT32 STL 应用代码被打包成名为“HT32_SafetyTest_Library_IEC60730_ClassB_vn_m.zip”的 zip 文件。

下载路径：https://mcu.holtek.com.tw/ht32/app.fw/SafetyTest_IEC60730_ClassB/

由于应用代码不包含固件库文件，用户需要在开始编译之前将解压缩的应用代码和固件库文件放到正确的路径中。应用代码文件包含两个文件夹，分别是“application”和“library”，其位置如图 3 所示。将这两个文件夹放到固件库根目录下，以完成文件路径配置，如图 4 所示。用户还可以将应用代码和固件库压缩文件解压到相同的路径中，也能达到相同的效果。在本例中，解压缩后在“application”文件夹下可以看到“SafetyTest_IEC60730_ClassB”目录。

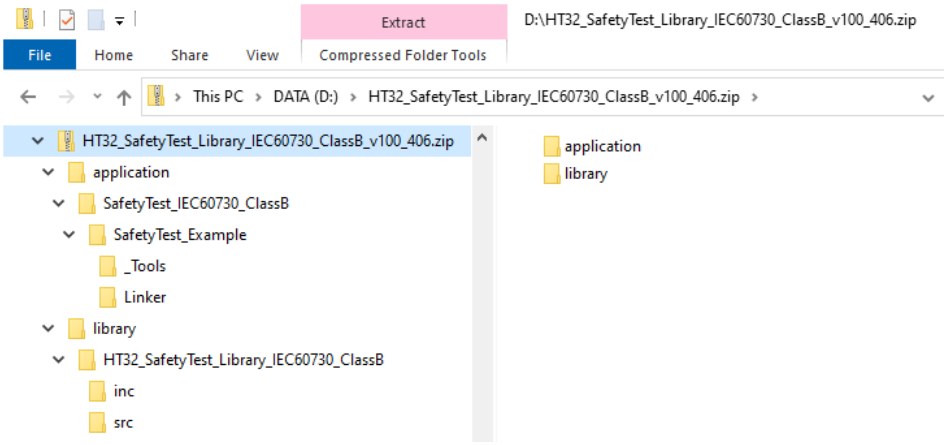


图 3. “HT32_SafetyTest_Library_IEC60730_ClassB_vn_m.zip”内容

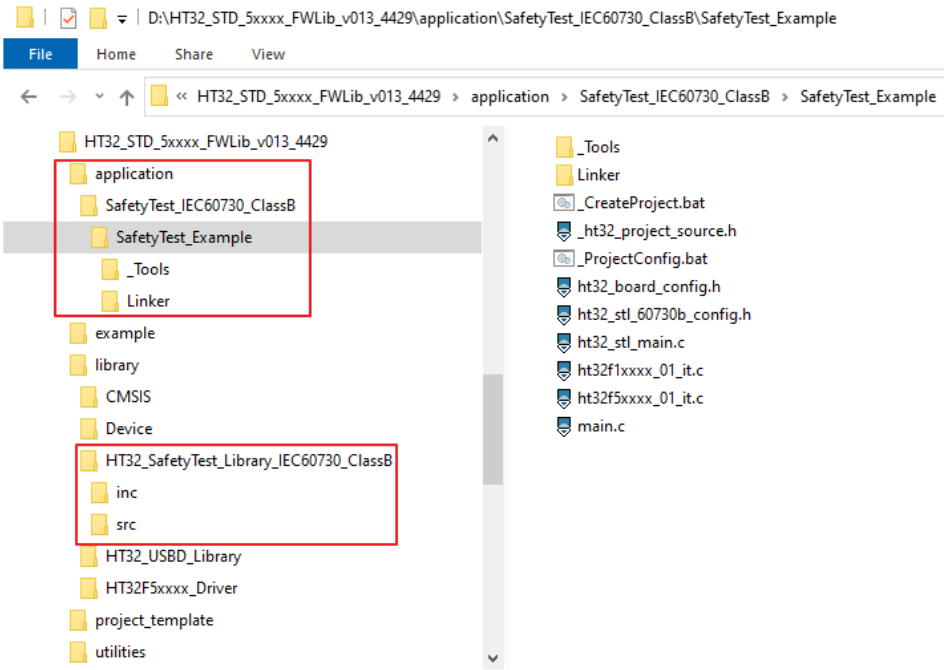


图 4. 解压后的文件内容

基本测试

以下步骤说明如何使用 Starter Kit 和 HT32 STL 进行基本测试，以确保硬件和软件环境都能正常工作。建议在产品中使用 HT32 STL 之前执行基本测试。

1. 使用 USB 数据线将 Starter Kit 连接到计算机上（详细描述请参考“硬件”章节）。
2. 解压固件库和 HT32 STL zip 文件，并正确配置文件路径（详细描述请参考“软件”章节）。
3. 通过执行“_CreateProject.bat”批处理文件启动创建项目流程。
   “\\application\SafetyTest_IEC60730_ClassB\SafetyTest_Example_CreateProject.bat”
4. 根据 Starter Kit 的设备型号打开相应的项目文件。例如，打开以下项目文件：
   “\\application\SafetyTest_IEC60730_ClassB\SafetyTest_Example\MDK_ARMv5\Project_50241.uvprojx”
5. 按下 Keil uVision 中的“Rebuild”图标（或按下 F7 热键），确认所有的构建操作都已成功，并且输出信息与下图相似。

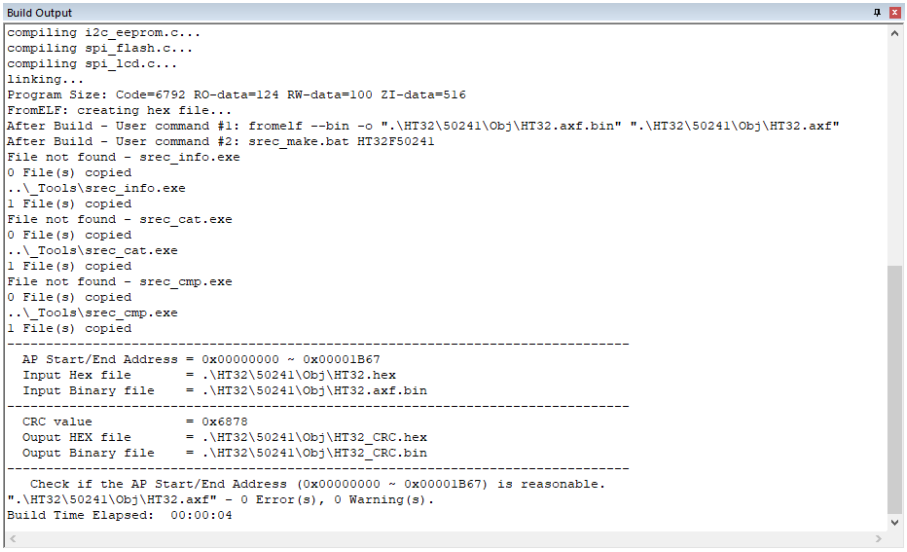


图 5. 生成代码后输出信息

6. 按下“LOAD”图标（或菜单中的“Flash→Download”选项）将图像下载到 Starter Kit 中。
7. 下载完成后，按下“reset”按钮，LED 将会闪烁（在左侧），表示硬件和软件环境运行正常。

系统架构

下面几节说明 HT32 STL 的系统架构，包括分层架构、文件/文件夹组织、安全停止函数、使用流程、存储器布局和全局变量冗余。

分层架构

下图显示了 HT32 STL 的分层架构，包括用户层、STL API 层和安全测试库（STL）层。分层结构可以帮助用户快速地将 HT32 STL 集成到他们的固件/产品中。STL API 层包括 STL 启动模块和主模块。

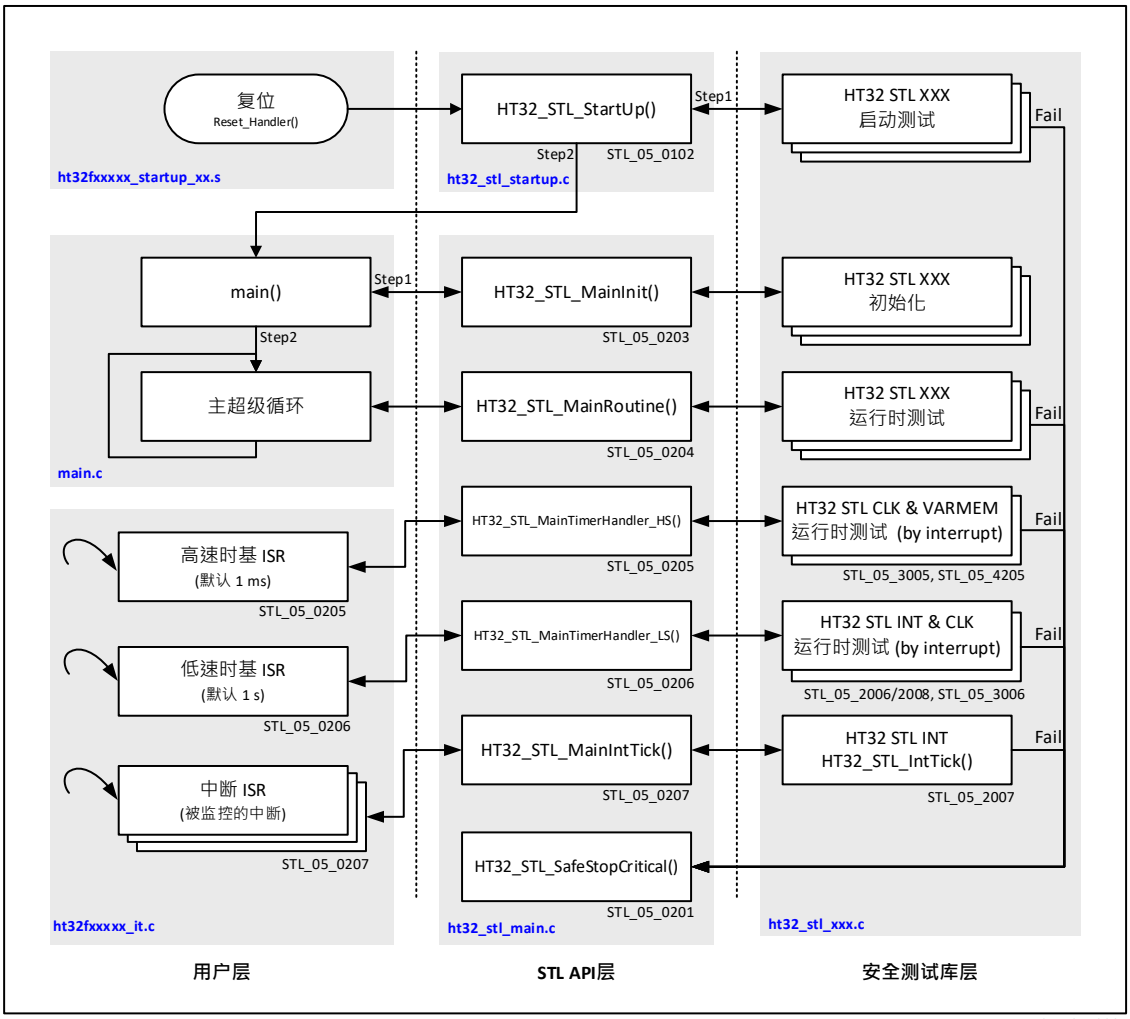


图 **6. HT32 STL 软件架构**

STL 启动模块用于对运行时无法测试的测试项执行启动测试。STL 主模块是帮助用户执行各项自检功能的主要部分。用户需要定期调用 STL 主模块的主程序函数。由于一些自检不能中断或需要不同时间之间的交互，主模块提供了高速和低速定时器处理程序来收集相关自检模块函数。用户需要在用户层配置高速和低速定时器，并调用 STL 主模块相应的定时器处理程序。

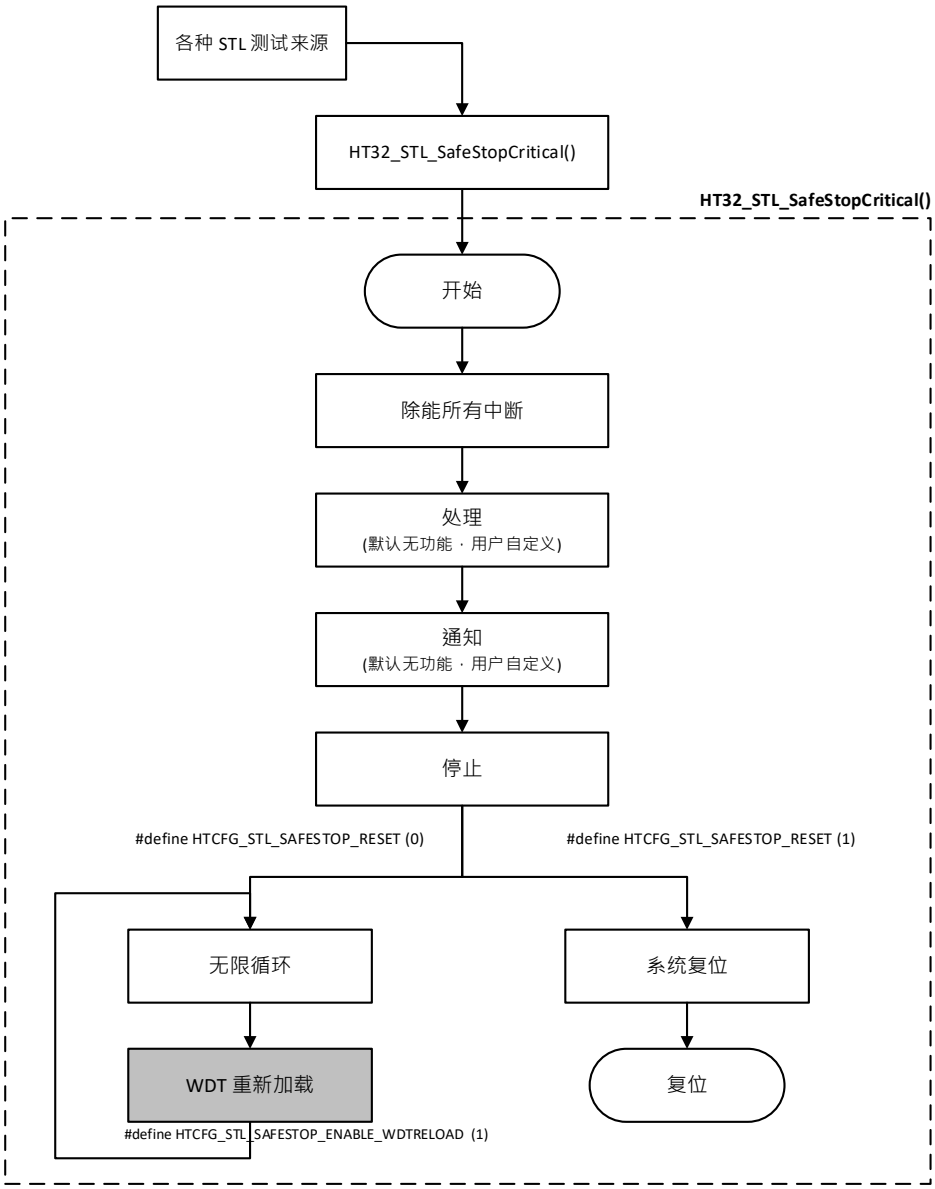
HT32 STL 组织

HT32 STL 应用代码包含两个文件夹，分别是“application”和“library”。下表列出了所有文件夹/文件及其描述。

文件夹/文件名称	描述
“\\library\HT32_SafetyTest_Library_IEC60730_ClassB\”	
inc\ht32_stl.h	HT32 STL 的主要头文件。
inc\ht32_stl_adc.h src\ht32_stl_adc.c	模拟 I/O（ADC）测试项的头文件和源代码。
inc\ht32_stl_clock.h src\ht32_stl_clock.c	时钟测试项的头文件和源代码。
inc\ht32_stl_cpu.h src\ht32_stl_cpu.c	CPU 测试项的头文件和源代码。
inc\ht32_stl_digitalio.h src\ht32_stl_digitalio.c	数字 I/O 测试项的头文件和源代码。
inc\ht32_stl_interrupt.h src\ht32_stl_interrupt.c	中断测试项的头文件和源代码。
inc\ht32_stl_invar_memory.h src\ht32_stl_invar_memory.c	非易失性存储器（Flash）测试项的头文件和源代码。
inc\ht32_stl_pc.h src\ht32_stl_pc.c	程序计数器测试项的头文件和源代码。
inc\ht32_stl_startup.h src\ht32_stl_startup.c	STL 启动模块的头文件和源代码。
inc\ht32_stl_var_memory.h src\ht32_stl_var_memory.c src\ht32_stl_var_memory_run.c	易失性存储器（SRAM）测试项的头文件和源代码。
inc\ht32_stl_debug.h	与调试相关的定义。
inc\ht32_stl_main.h	STL 主模块的头文件。
“\\application\SafetyTest_IEC60730_ClassB\SafetyTest_Example\”	
_Tools\ srec_cat.exe srec_cmp.exe srec_info.exe srec_make.bat srec_make_IAP.bat srec_make_manual.bat	SRecord 的相关工具和脚本，可以帮助用户计算/修改非易失性存储器（Flash）测试的图像校验和。
linker\linker.lin	不同存储器布局配置的 Keil 链接器脚本范例。
_CreateProject.bat	创建项目文件的批处理脚本。
_ht32_project_source.h	项目源文件，使用“#include”方式将源代码文件添加到项目编译列表中。此文件包含在“man.c”中。
_ProjectConfig.bat	创建项目文件的配置。
ht32_board_config.h	用于引脚分配和其它设置的板相关配置文件。
ht32_stl_60730b_config.h	HT32 STL 配置文件。
ht32_stl_main.c	HT32 STL 主模块的源代码文件。
ht32f1xxxx_01_it.c ht32f5xxxx_01_it.c	包含中断服务程序的文件。
main.c	主函数的源代码文件。

安全停止函数

因为用户需要根据应用的需求来修改程序，所以 HT32 STL 安全停止函数位于 STL 主模块中，而不是位于安全测试库层中。处理、通知和停止这三个主要步骤可以帮助用户将系统停止在安全状态。用户必须正确执行安全停止函数，才能确保其按预期运行。如果检测到任何故障，将调用 HT32 STL 安全停止函数。



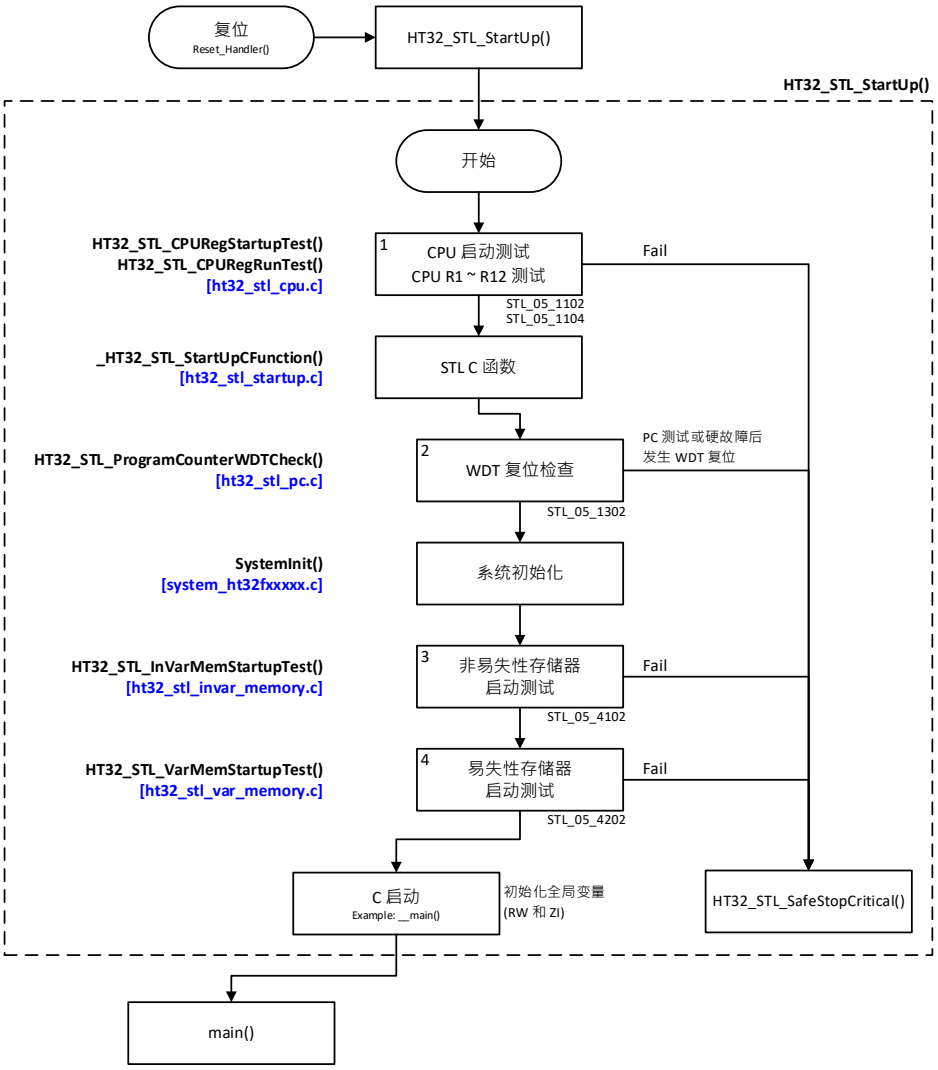
注：以下定义位于 “ht32_stl_60730b_config.h” 中
#define HTCFG_STL_SAFESTOP_RESET (0)
#define HTCFG_STL_SAFESTOP_ENABLE_WDTRELOAD (1)

STL_05_0201

图 7. 安全停止函数流程图

使用流程 – 启动程序

在 MCU 复位后，STL 启动函数由“ht32fxxxxx_startup_nn.s”调用。它会调用 CPU、非易失性存储器和易失性存储器的启动测试函数。此外，还会检查 WDT 复位，以确保在程序计数器测试失败（由 WDT 复位引起）后调用 HT32 安全停止函数。



注：测试项 1、2、3 和 4 位于安全测试库层的 “ht32_stl_xxx.c” 中。 STL_05_0102

图 8. 启动程序流程图

使用流程 – 主初始化程序

主初始化函数将在进入“main()”函数后调用一次。它会对 STL 主全局变量冗余进行初始化，实现 MCU 和各自检模块的必要配置。

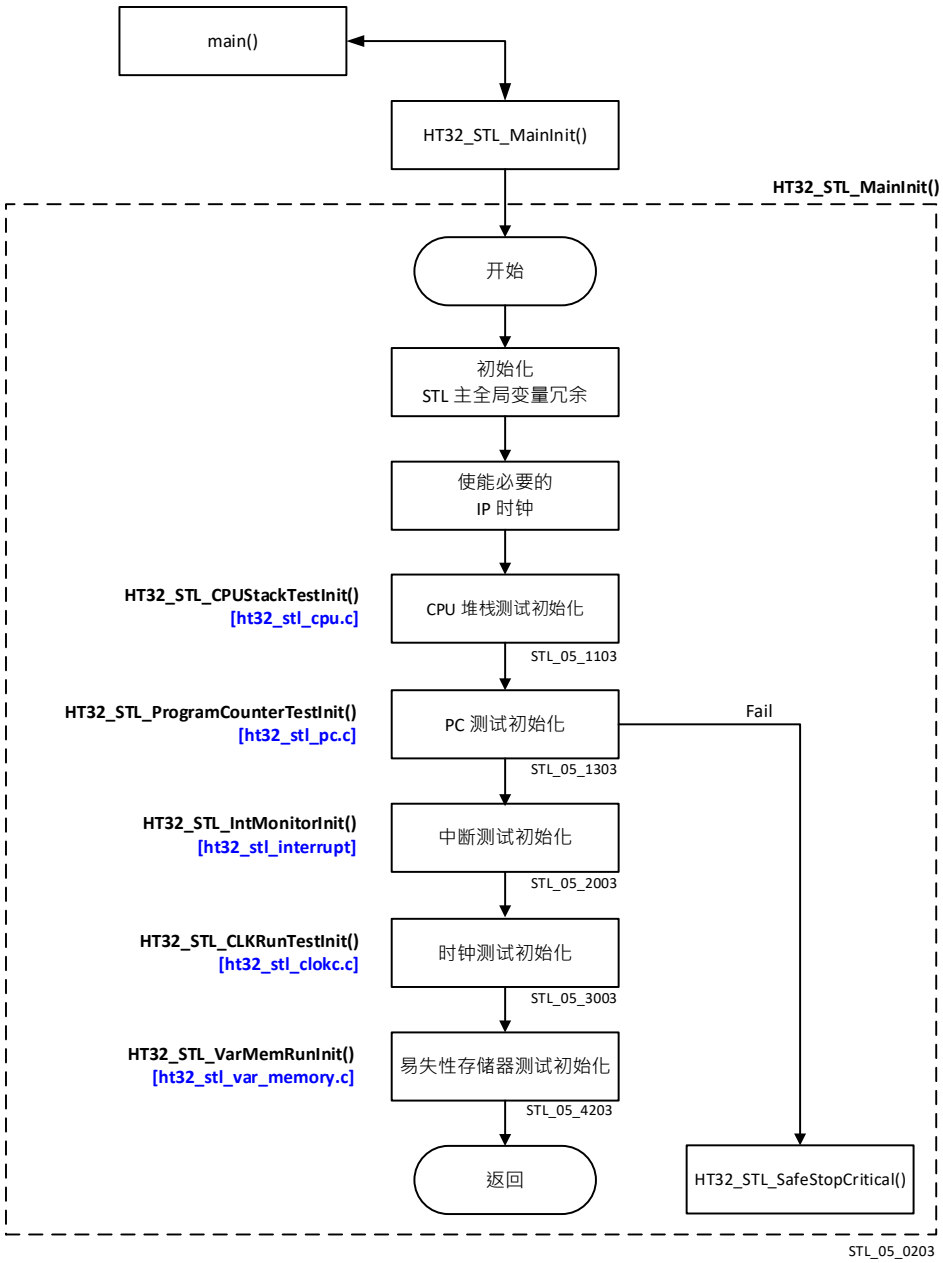


图 9. 主初始化程序流程图

使用流程 – 主程序

主程序函数在进入“main()”函数后由用户的主超循环定期调用，可以帮助用户执行各项自检功能，包括 CPU、程序计数器、中断、非易失性存储器、GPIO（可选）和模拟 I/O（ADC）。

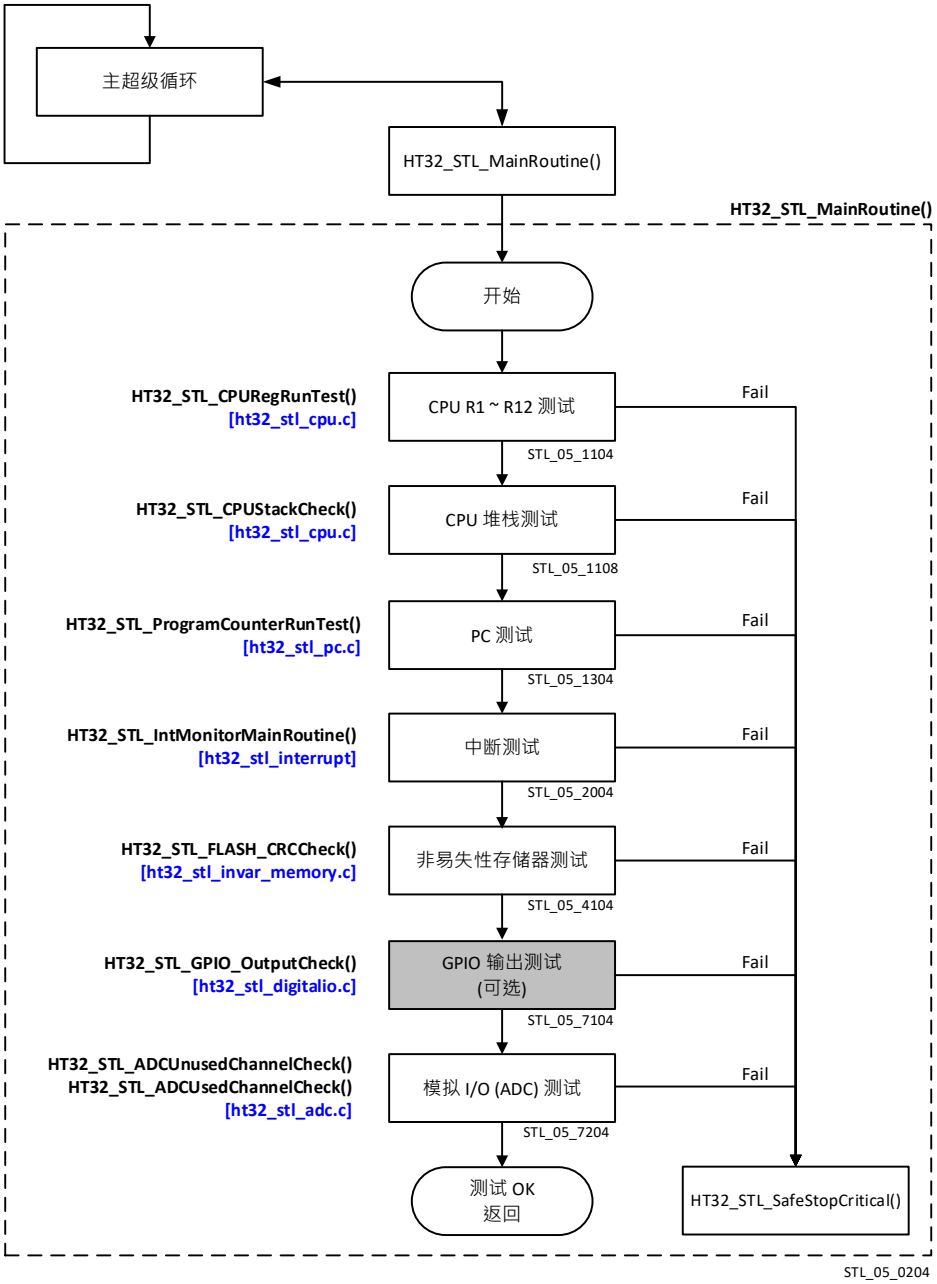


图 10. 主程序流程图

使用流程 – 中断服务程序（ISR）

高速定时器处理程序会调用时钟和易失性存储器的自检功能。用户需要在用户层配置高速定时器，在高速 ISR 中调用 STL 主模块的高速定时器处理程序。

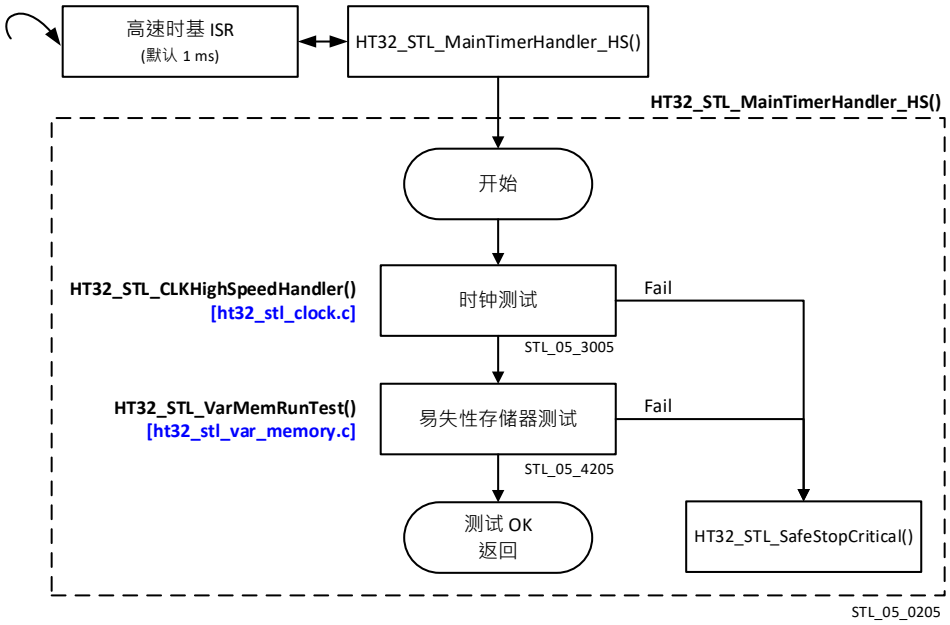


图 11. 高速中断服务程序流程图

低速定时器处理程序会调用中断和时钟自检功能。用户需要在用户层配置低速定时器，在低速 ISR 中调用 STL 主模块的低速定时器处理程序。

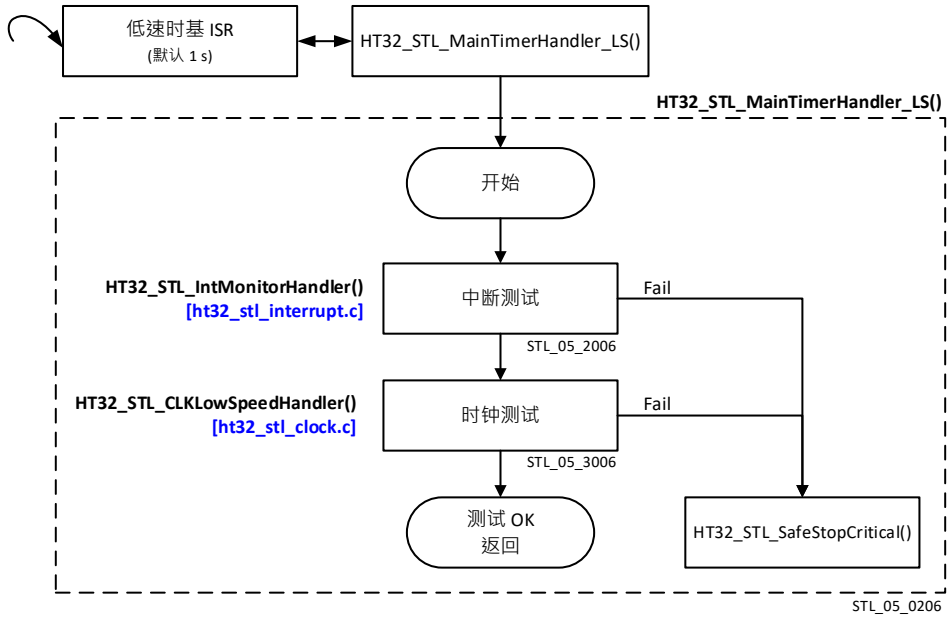


图 12. 低速中断服务程序流程图

主中断节拍函数是为 HT32 STL 中断监控机制而设计的。被监控中断的中断服务程序（ISR）必须调用主中断节拍函数以执行中断监控机制的相关进程。

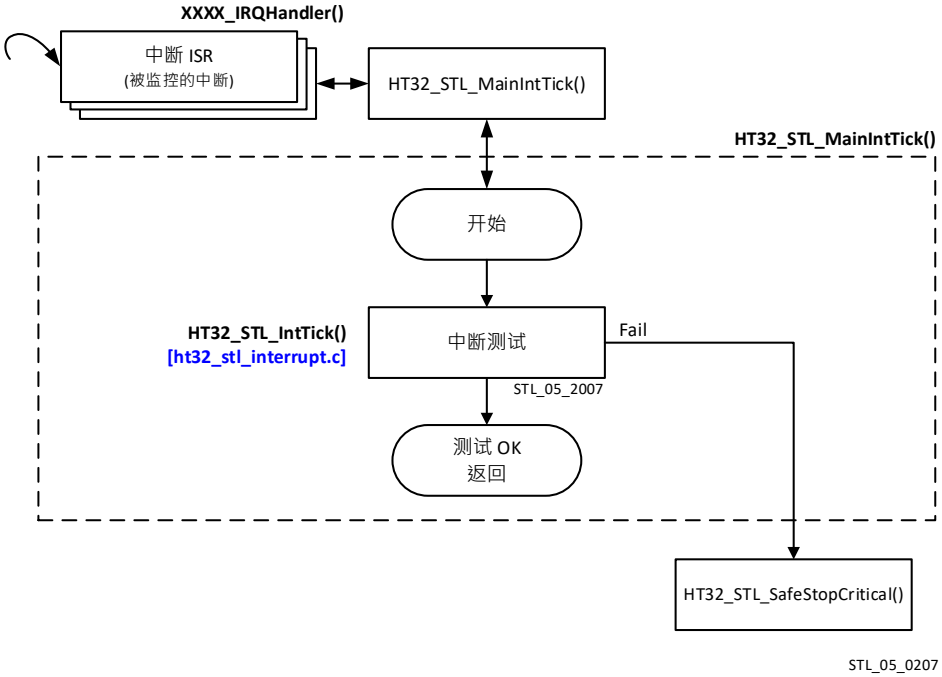


图 13. 主中断节拍函数流程图

存储器布局

下图显示了系统易失性存储器（SRAM）的默认存储器布局。“#define XXXX”是可以在“ht32_stl_60730b_config.h”配置文件中找到的配置设置。这种存储器布局要求用户更改 RW 基址，是实现存储器分区的一种方便的方法。用户还可以使用链接器脚本来实现更复杂的布局，例如将其它 Class A 变量（与 Class B 功能无关）定位到另一个空间，减少冗余存储器的大小，从而节省存储器的使用。Keil 链接器脚本范例可以在以下路径中找到。详细描述请参考“不同的存储器布局”章节。

“\\application\SafetyTest_IEC60730_ClassB\SafetyTest_Example\Linker\linker.lin”

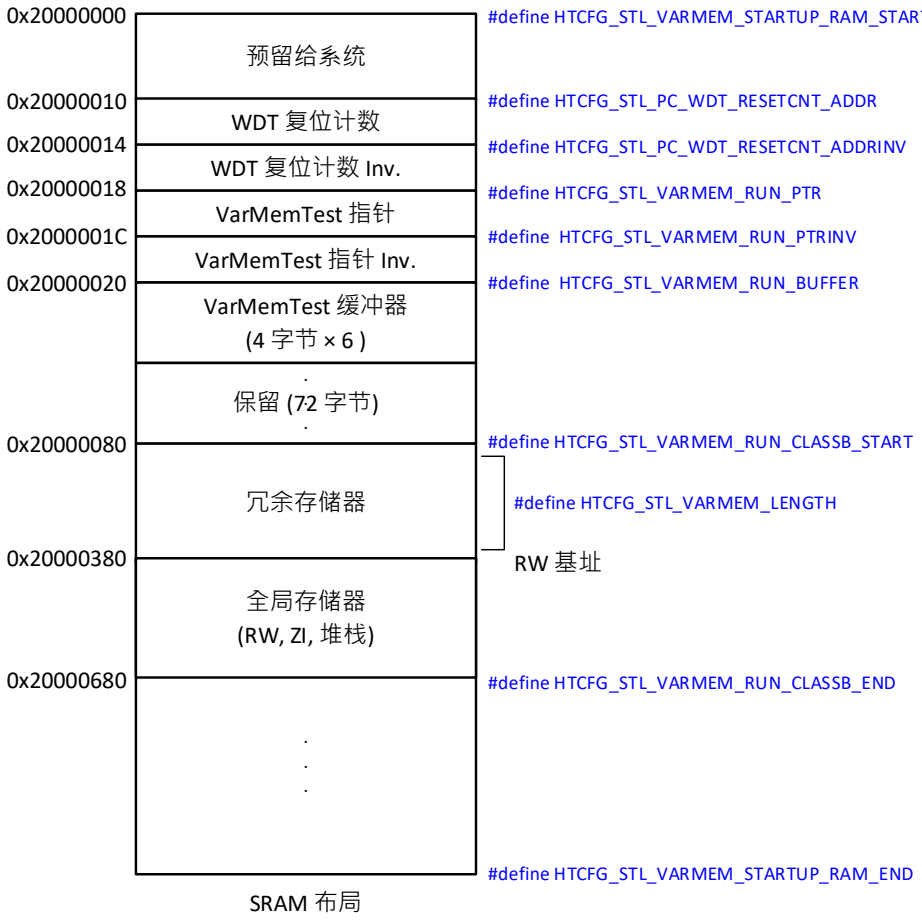


图 14. HT32 STL 默认存储器布局

下图是一个简化的存储器布局，有助于理解易失性存储器（SRAM）测试项的测试范围。易失性存储器的启动测试从“HTCFG_STL_VARMEM_STARTUP_RAM_START”地址开始，到“HTCFG_STL_VARMEM_STARTUP_RAM_END”地址结束。易失性存储器（包括 Class B 全局存储器和冗余存储器）运行时自检从“HTCFG_STL_VARMEM_RUN_CLASSB_START”地址开始，到“HTCFG_STL_VARMEM_RUN_CLASSB_END”地址结束。

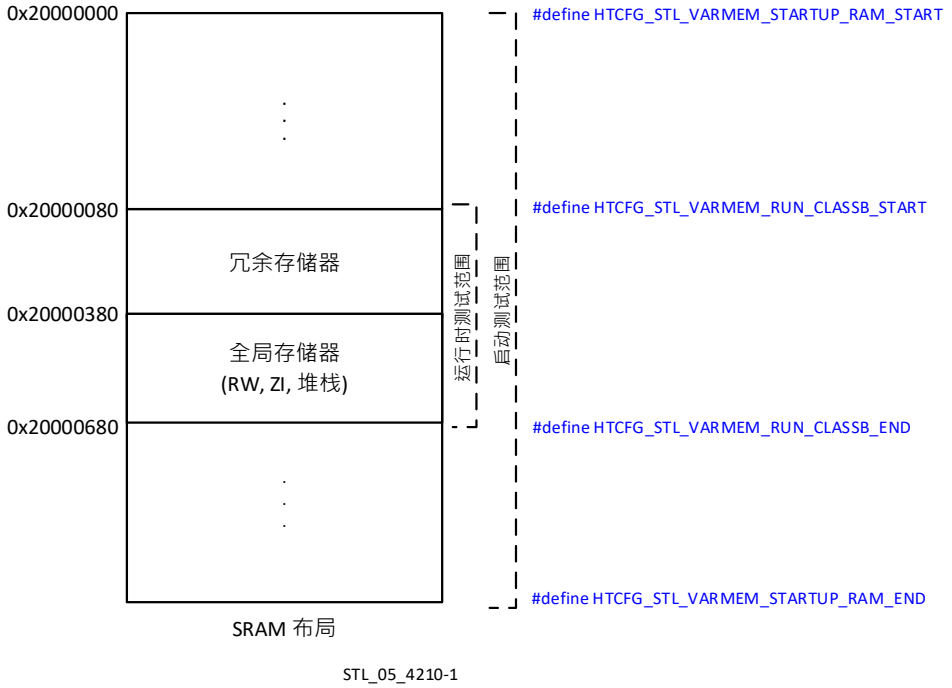


图 15. 易失性存储器的测试范围

Class B 全局变量冗余

为了提高存储在 SRAM 中信息的可靠性，全局变量应该以不同的格式保存在被物理分区的双存储器中。如前一节的存储器映射所示，系统为 Class B 全局变量保留了一个冗余存储器区域。与安全功能相关的 Class B 全局变量应该使用该区域来保存冗余值。该机制只是通过用全局变量地址减去偏移量（由 HTCFG_STL_VARMEM_LENGTH 定义），将存储器地址从全局存储器重映射到冗余存储器。例如，如果全局变量的地址是 0x20000390，那么冗余地址应该是 0x20000390 - 0x300 = 0x20000090。该机制使用取反运算作为保存格式。

要求	方法	范例
独立区域	保留一个冗余存储器区域并将全局变量重映射于其中。	地址 全局变量：0x20000390 冗余：0x20000090
不同格式	取反运算	全局变量：0x000055AA 冗余：0xFFFFAA55 冗余 = ~(全局变量)

根据上述要求，该机制提供了以下宏来实现冗余流程。

宏	说明	范例
SYNC	读取全局变量，取反并保存到冗余。	A' = ~A
SET	将全局变量和冗余设置为相同的值。	A = A' = value
SAVE	读取冗余，取反并保存到冗余。	A' = ~A'
CHECK	通过 XOR 运算检查全局变量和冗余是否正确。	Is (A ^ A') == 0xFFFFFFFF ?

VERIFY	与 CHECK 宏的操作相同，但如果检查失败，需进入安全停止函数。	If (A ^ A') == 0xFFFFFFFF
		  return A
		Else
		  call Safe Stop function

注：A'表示变量 A 的冗余。

下面的范例说明如何使用这些宏来实现与 Class B 安全相关的全局变量冗余。注意，全局变量包括 RW（有初始值）、ZI（初始值为零，无初始值时会被设置为“0”）和 Stack，由 HT32 STL 启动函数调用的 C 启动函数初始化。详细描述请参考 STL_05_0102 启动函数。此外，自检模块内部的（全局变量的）冗余通常由 HT32 STL XXX 初始化函数发起，该函数应该在进入“main()”函数（调用一次）之初由 HT32 STL 主初始化函数调用。反之，位于 STL 主模块层或用户层的冗余应该由用户使用此处提到的宏来处理。

```
u32 gINT_TestFlag = 0x00000055;
....
SYNC(gINT_TestFlag); // Sync once when init, (gINT_TestFlag')= 0xFFFFFFAA;
....
SET(gINT_TestFlag) = 0x00000001; // gINT_TestFlag = gINT_TestFlag' = 0x1
SAVE(gINT_TestFlag); //(gINT_TestFlag') = ~(gINT_TestFlag')
....
if (CHECK(gINT_TestFlag)) // Execute following code only if the redundant is OK.
{
  if (gINT_TestFlag == 0x00000001)
  {
    // Do process here
  }
}
....
if (VERIFY(gINT_TestFlag) == 0x00000001) // Enter the Safe Stop function when CHECK failed.
{
  // Do process here
}
```

使用说明

集成范例与检查点

第一步是将 HT32 STL 集成到项目文件中。将下列文件夹和文件从 HT32 STL 范例路径复制到项目根目录下（与“main.c”的路径相同）。

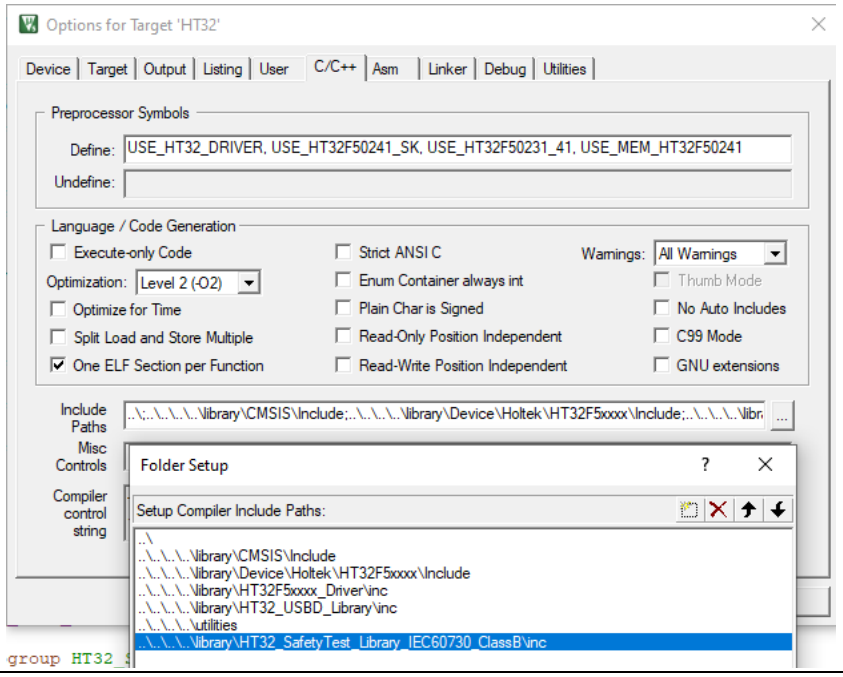
“\\application\SafetyTest_IEC60730_ClassB\SafetyTest_Example\”

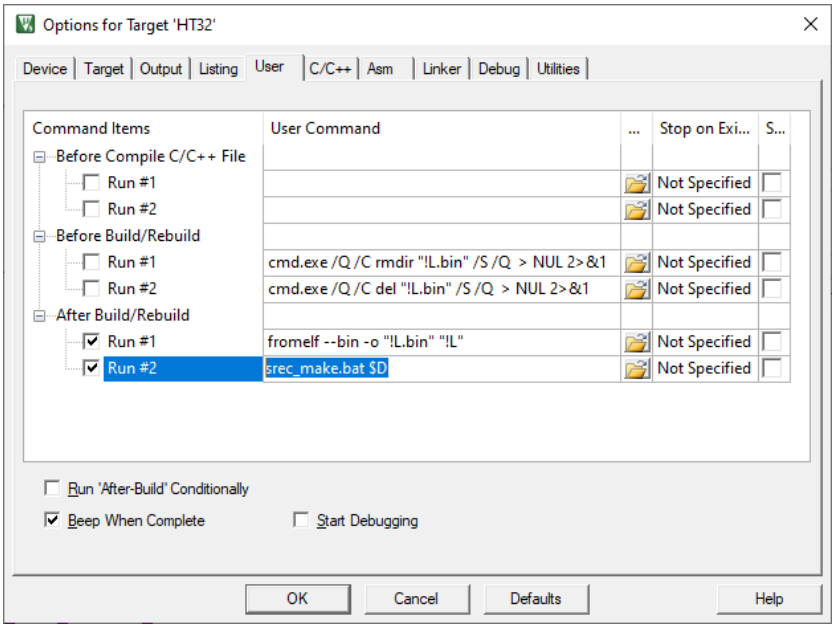
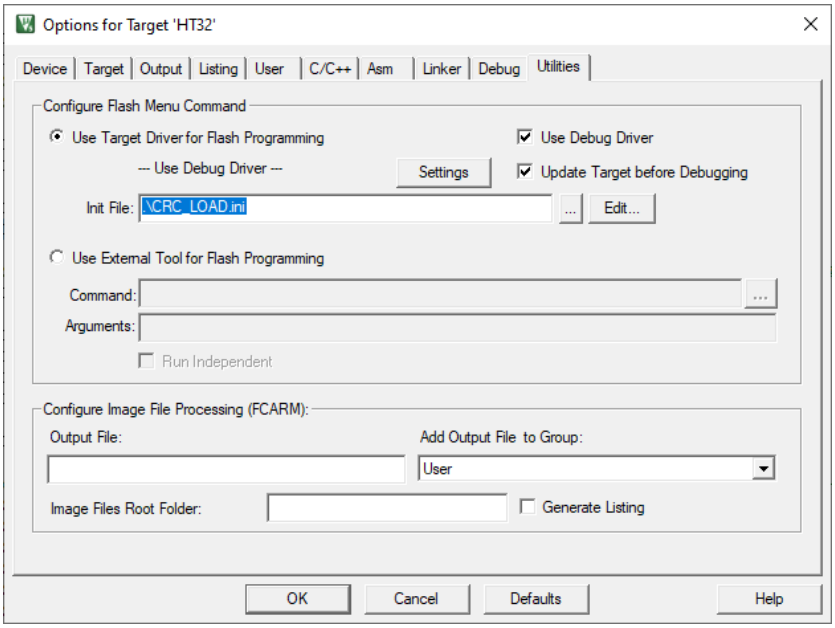
“_Tools\”
“Linker”
“_ht32_project_source.h”
“ht32_stl_60730b_config.h”
“ht32_stl_main.c”

还需将 HT32 STL 库文件定位到以下路径（请参考“HT32 STL 组织”章节）。

“\\library\HT32_SafetyTest_Library_IEC60730_ClassB\inc”
“\\library\HT32_SafetyTest_Library_IEC60730_ClassB\src”

为了满足 HT32 STL 的需求，下表列出了修改项目和源代码的步骤。

步骤	相关文件	说明
1	Include Path	将下面的 Include Path 添加到项目的“Options for Target”中。 “..\..\..\..\library\HT32_SafetyTest_Library_IEC60730_ClassB\inc” 

步骤	相关文件	说明
2	After Build	添加 After Build 命令“srec_make.bat $D”。  **需要将“srec_make.bat”文件从“_Tools”文件夹手动复制到项目文件文件夹（例如“MDK_ARMv5\”）中。**
3	RW Base	将 RW 基址由 0x20000000 修改为 0x20000380（默认值）。请参考“存储器布局”和“HT32 STL 设置”章节来修改 RW 基址。
4	Flash Programming Init File	将下面的“CRC_LOAD.ini”文件添加到 Keil uVision 的 Flash 烧录初始化文件中。该文件是在生成代码之后由“srec_make.bat”创建的（意味着还需在此手动输入文件名，因为完成设置并生成代码是不完整的）。“CRC_LOAD.ini”文件用于将 CRC 校验和烧录到 Flash 中，以进行非易失性存储器测试。 
5	WDT	在“system_ht32fxxxxx_nn.c”文件中，将“WDT_ENABLE”由 0 修改到 1，并更改 WDT 相关设置。需要将重新加载 WDT 函数添加到代码中。

步骤	相关文件	说明
6	Timer	在使用 HT32 STL 之前，需要正确配置高速和低速定时器。建议参考“基本测试”章节中 HT32 STL 测试范例代码的“main.c”和“ht32fxxxxx_nn_it.c”文件。 高速定时器（默认 1 ms） 低速定时器（默认 1 s）
7	IRQ Handler	需要注意以下 IRQ 处理程序。请参考 HT32 STL 范例代码中的“ht32fxxxxx_nn_it.c”文件，仔细比较/修改到相应的文件。 “HardFault_Handler()” “SysTick_Handler()” “RTC_IRQHandler()”
8	“ht32fxxxxx_nn_it.c”	需要添加两个 include 文件。 #include "ht32_stl_60730b_config.h" #include "ht32_stl.h"
9	“main.c”	需要添加三个 include 文件。 #include "ht32_stl_60730b_config.h" #include "ht32_stl.h" #include "_ht32_project_source.h"

以上操作完成后，需要修改项目的额外系列文件。下图和表格说明了如何将 HT32 集成到用户固件中的步骤。同时也是一个检查列表，有助于确认集成是否正确。

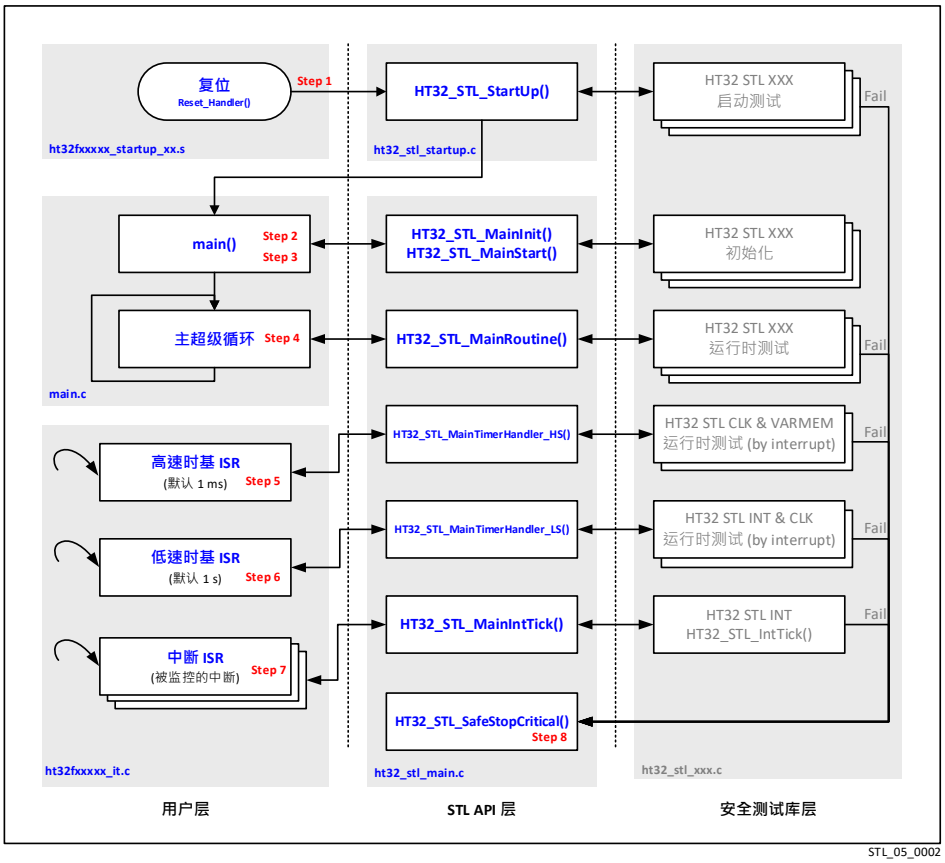


图 16. 集合范例和检查点

步骤	相关文件	函数和说明

1	“ht32fxxxxx_startup_xx.s”	**HT32_STL_StartUp()** (Note 1)
		此函数必须由“Reset_Handler()”调用。
		startup_ht32f5xxxx_02.s
		ENDIF
		DCD _RESERVED
		DCD UART0_IRQHandler
		DCD UART1_IRQHandler
		; Reset handler routine
		Reset_Handler PROC
		EXPORT Reset_Handler
		IMPORT HT32_STL_StartUp
		IMPORT __main
		LDR R0, =BootProcess
		BLX R0
		LDR R0, =HT32_STL_StartUp
		BLX R0
		LDR R0, =__main
		BX R0

2	“main.c”	**HT32_STL_MainInit()** 此函数必须在进入“main()”函数后调用（一次）。 main.c int main(void) { HT32_STL_MainInit(); //...
3	“main.c”	**HT32_STL_MainStart()** 此函数必须在进入“main()”函数后（高/低速定时器初始化之后）调用（一次）。 main.c int main(void) { HT32_STL_MainInit(); //... RTC_Configuration(); SysTick_Configuration(); //... HT32_STL_MainStart();

4	“main.c”	**HT32_STL_MainRoutione()**
		此函数必须由“main()”函数中的用户超级循环定期调用。
		main.c
		int main(void)
		{
		//...
		while (1)
		{
		HT32_STL_MainRoutine();
		}

5	“ht32fxxxxx_nn_it.c”
	(Note 2)
		HT32_STL_MainTimerHandler_HS()
		此函数必须由高速定时器 ISR 调用。默认时间为 1 ms。
		ht32f5xxxx_01_it.c
		void SysTick_Handler(void)
		{
		static u32 gTimerCount = 0;
		HT32_STL_MainTimerHandler_HS();

6	“ht32fxxxxx_nn_it.c”
	(Note 2)
		HT32_STL_MainTimerHandler_LS()
		This function must be called by the Low-Speed Timer ISR. Default time is 1 second.
		此函数必须由低速定时器 ISR 调用。默认时间为 1 s。
		ht32f5xxxx_01_it.c
		void RTC_IRQHandler(void)
		{
		u32 uFlags = RTC_GetFlagStatus();
		if (uFlags & RTC_FLAG_CSEC)
		{
		HT32_STL_MainTimerHandler_LS();

步骤	相关文件	函数和说明

7	“ht32fxxxxx_nn_it.c” (Note 2)	**HT32_STL_MainIntTick()**
		此函数必须由被监控中断的 ISR 调用。“&gIntMonitorList[n]”参数是带有中断 ID 的结构指针。此外，还需编辑在安全停止 ID 中的中断 ID（“n”和 STL_DID_INT_TICK 进行按位或运算，用于调试目的）。
		HT32_STL_SAFESTOP_FUN(STL_DID_INT_TICK | **n**);
		详细描述请参考“gIntMonitorList[]”（在“ht32_stl_main.c”中）以及步骤 8。
		ht32f5xxxx_01_it.c
		`void BFTM0_IRQHandler(void)`
		`{`
		`  BFTM_ClearFlag(HT_BFTM0);`
		`  #if (HTCFG_STL_RUNTEST_EN_INT == 1)`
		`  {`
		`    extern STL_IntMonitorList_Typedef gIntMonitorList[];`
		`    if (HT32_STL_MainIntTick(&gIntMonitorList[1]) == ERROR)`
		`    {`
		`      __STL_ERR_DBGprintf("\r\nISR FAIL: HT32_STL_MainIntTi`
		`      HT32_STL_SAFESTOP_FUN(STL_DID_INT_TICK | 1);`
		`    }`

8	ht32_stl_main.c	**gIntMonitorList[]**
		中断信息列表保存了中断测试的参数。根据应用中需要监控的中断来编辑结构体并添加新值。
		ht32_stl_main.c
		`#if (HTCFG_STL_RUNTEST_EN_INT == 1)`
		`#define INT_INFO_CNT (3)`
		`STL_IntMonitorList_Typedef gIntMonitorList[INT_INFO_CNT] =`
		`{`
		`// !!!Notice !!!`
		`// 1. All the variable of "gIntMonitorList[]" using redundant method by the STL Layer, "gIntMonitorList" should not modify on the fly.`
		`// 2. "uStatus" can be changed by "HT32_STL_IntMonitorCHCmd()".`
		`// uType, uStatus, uReset, uCounter, uMax, uMin`
		`// {STL_INT_FASTER, ENABLE, FALSE, 0, 1100 , 900 }, // Note Here`
		`// {STL_INT_SLOWER, DISABLE, FALSE, 0, 1100 , 900 }, // Note Here`
		`  {STL_INT_FASTER, ENABLE, FALSE, 0, 1000 + 200 , 1000 - 200 }, // Systick, 1 ms`
		`  {STL_INT_FASTER, ENABLE, FALSE, 0, 100 + 20 , 100 - 20 }, // BFTM0, 10 ms`
		`// {STL_INT_SLOWER, ENABLE , FALSE, 0, 6 , 4 }, // BFTM1, 5 S`
		`};`
		`#endif`

9	ht32_stl_main.c	**HT32_STL_SafeStopCritical()** 确认安全停止函数操作正确，运行正常。

注：1. 对于 HT32 STL 的默认范例，“ht32fxxxxx_startup_xx.s”由“_CreateProject.bat”自动修改。使用“gsar.exe”工具，相关配置位于“_ProjectConfig.bat”文件中。如果使用现有的项目/代码，可能需要手动将“HT32_STL_StartUp()”函数修改到启动文件中。

2. 用户可以将中断服务程序（ISR）从“ht32fxxxxx_nn_it.c”中移到另一个 C 源代码文件中。需要确认“HT32_STL_MainTimerHandler_HS()”、“HT32_STL_MainTimerHandler_LS()”和“HT32_STL_MainIntTick()”函数在相应的定时器 ISR 中已被正确调用。

HT32 STL 设置

下表显示了位于“ht32_stl_60730b_config.h”文件中的一些 HT32 STL 设置（可以在“\\application\SafetyTest_IEC60730_ClassB\SafetyTest_Example”范例的根目录中找到）。这些设置以“HTCFG_STL_”前缀开始，修改前请检查每个设置的描述。

设置的名称	描述
安全停止	
HT32_STL_SAFESTOP_FUN(id)	安全停止函数的别名。默认值是“HT32_STL_SafeStopCritical(id)”。有助于用户快速更改安全停止的实现。
HTCFG_STL_SAFESTOP_LED_INDICATOR	1：进入安全停止函数时打开 LED。 0：无操作。
HTCFG_STL_SAFESTOP_RESET	在安全停止函数中复位系统。
HTCFG_STL_SAFESTOP_ENABLE_WDTRELOAD	在安全停止函数中重新加载 WDT。当“HTCFG_STL_SAFESTOP_RESET = 1”时，此设置无意义。
HTCFG_STL_SAFESTOP_WHEN_HARDFAULT	如果发生硬故障异常，将进入安全停止函数。
STL 测试控制	
HTCFG_STL_STARTUPTEST_EN_CPU HTCFG_STL_STARTUPTEST_EN_INVARMEM HTCFG_STL_STARTUPTEST_EN_VARMEM HTCFG_STL_RUNTEST_EN_CPU HTCFG_STL_RUNTEST_EN_PC HTCFG_STL_RUNTEST_EN_INT HTCFG_STL_RUNTEST_EN_CLK HTCFG_STL_RUNTEST_EN_INVARMEM HTCFG_STL_RUNTEST_EN_VARMEM HTCFG_STL_RUNTEST_EN_DIO HTCFG_STL_RUNTEST_EN_AIO	这些设置用于使能和除能每个测试函数。当设置为 0 时，将忽略测试函数。
程序计数器	
HTCFG_STL_PC_FUN1ADDR	用于初始化测试函数 1 的 Flash 存储器地址并执行该测试函数。
HTCFG_STL_PC_FUN2ADDR	用于初始化测试函数 2 的 Flash 地址并执行该测试函数。
HTCFG_STL_PC_WDT_NORMALRESETCNT_SAFESTOP	正常模式下最大 WDT 复位计数值。当 WDT 复位发生且复位计数值达到正常模式下最大 WDT 复位计数值时，STL 将进入安全停止函数。正常模式意味着 WDT 复位原因不是程序计数器测试失败或硬故障。
中断	
HTCFG_STL_INT_MAINROUTINE_MAXCHKTIME	最大中断主程序检查时间，应根据主超级循环的总执行时间进行调整。如果全局计数值“gINT_MainRoutineCount”大于最大检查时间（意味着在一段时间内不调用中断监控处理程序），则会执行安全停止函数。
时钟	
HTCFG_STL_CLK_HIGHSPEED_HZ	高速中断的频率。
HTCFG_STL_CLK_LOWSPEED_HZ	低速中断的频率。
HTCFG_STL_CLK_TOLERANCE	时钟容差，例如“0.25”。
非易失性存储器（Flash）	
HTCFG_STL_INVARMEM_HWCRC	1：硬件 CRC，0：软件 CRC。
HTCFG_STL_INVARMEM_BLOCK_SIZE	每次调用“HT32_STL_FLASH_CRCCheck()”时的块大小。
HTCFG_STL_INVARMEM_AP_FLASH_START	应用图像的起始地址。
易失性存储器（SRAM）	

设置的名称	描述
HTCFG_STL_VARMEM_STARTUP_RAM_START	启动测试的起始地址。
HTCFG_STL_VARMEM_STARTUP_RAM_END	启动测试的结束地址。必须是 16 字节对齐。
HTCFG_STL_VARMEM_RUN_CLASSB_START	运行时测试的起始地址。
HTCFG_STL_VARMEM_RUN_CLASSB_END	运行时测试的结束地址。
HFCFG_STL_VARMEM_RUN_BLOCK_SIZE	运行时测试的块大小。必须为 2~x（x 是 VarMemTest 缓冲器大小的 1/4）。
HFCFG_STL_VARMEM_RUN_BLOCK_OVERLAP	运行时测试的重叠。
HTCFG_STL_VARMEM_RUN_TEST_COUNT_MS	易失性存储器测试的测试时间间隔（以微秒为单位）。
HTCFG_STL_VARMEM_MARCHX	1：March X，0：March C。March X 比 March C 快。
HTCFG_STL_VARMEM_RUN_PTR	运行时测试的存储器指针。
HTCFG_STL_VARMEM_RUN_PTRINV	存储器指针的冗余。
HTCFG_STL_VARMEM_RUN_BUFFER	运行时测试的备份缓冲器。
HTCFG_STL_VARMEM_LENGTH	Class B/冗余区域的长度。
模拟 I/O	
HTCFG_STL_ADC_INTERCH	ADC 内部通道编号。
HTCFG_STL_ADC_INTERCH_MAX	ADC 最大期望值。
HTCFG_STL_ADC_INTERCH_MIN	ADC 最小期望值。
HTCFG_STL_ADC_TIMEOUT_LOOP	“HT32_STL_ADCInternalChannelCheck()”函数的超时循环计数。应根据 ADC 设置的转换时间进行调整。

不同的存储器布局

为了将其它 Class A 变量（与 Class B 函数无关）定位到另一个空间，以减少冗余存储器的大小和易失性存储器运行时测试的测试时间，下面的路径中提供了一个 Keil 链接器脚本范例。

“\\application\SafetyTest_IEC60730_ClassB\SafetyTest_Example\Linker\linker.lin”

链接器脚本有助于实现复杂的存储器布局。下图显示了使用链接器脚本时的存储器布局范例。

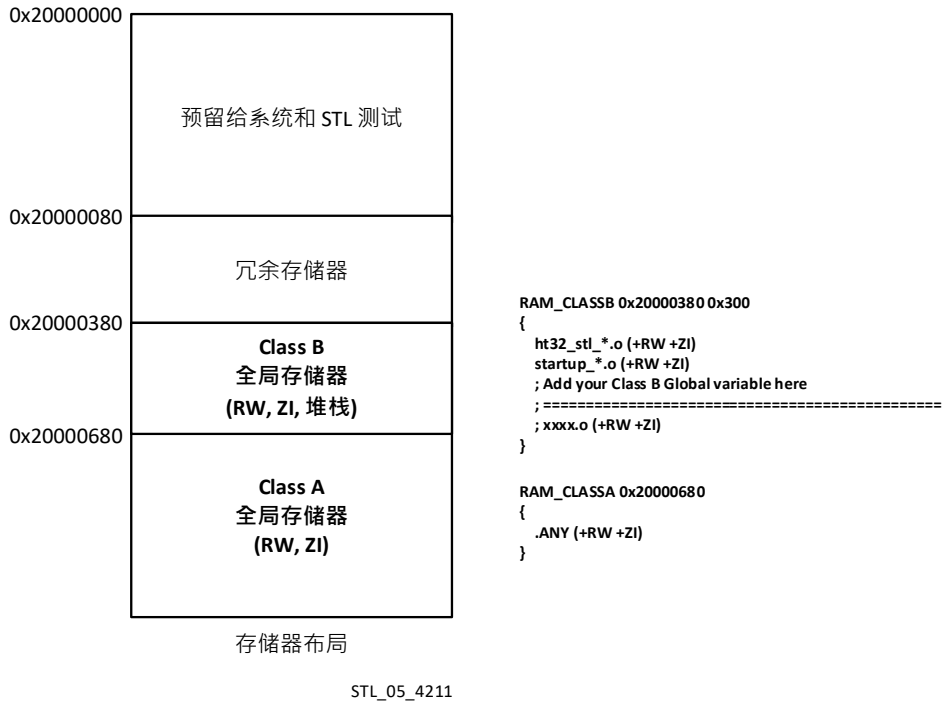


图 17. 使用链接器脚本的存储器布局

请参考下面的步骤在项目中使用链接器脚本。

1. 打开 Keil uVision 中项目的“Options for Target”，切换到“Linker”选项卡，将“Scatter File”设置为“linker.lin”（在前面的小节中有提到文件路径）。

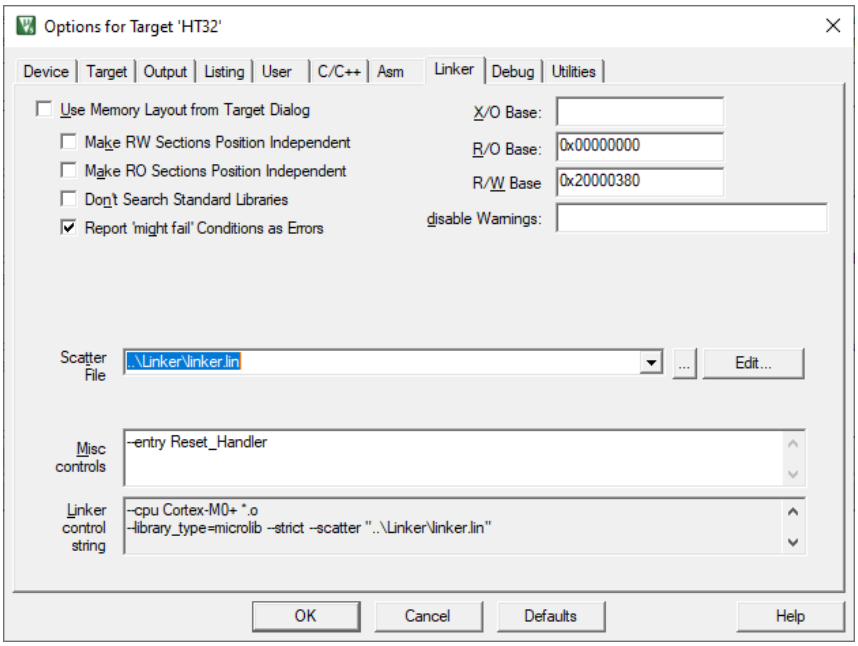


图 18. 使用链接器脚本（linker.lin）的链接器设置

2. 打开“_ht32_project_source.h”并将“HTCFG_ENABLE_PROJECT_SOURCE”设置由 1 修改为 0。将以下文件添加到项目编译列表中，如下所示。

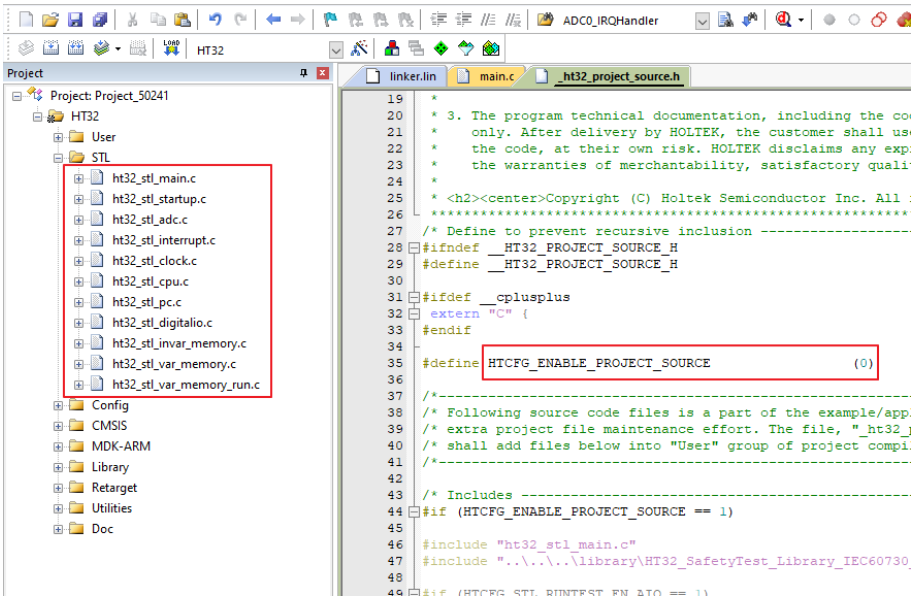


图 19. 使用链接器脚本时的设置和编译列表

HT32 STL 花费时间

在项目中添加 HT32 STL 之后，需要考虑的一个重要事项是每个 HT32 STL 测试项的花费时间，包括启动测试和定期自检。测试花费时间影响应用的实现和性能。例如，易失性存储器测试项在测试期间访问 RAM（将备份 RAM 内容并恢复），需要最高的中断优先级，并且不能被其它函数中断。了解易失性存储器测试项的花费时间有助于配置适当的 HT32 STL 参数以及如何定位代码和 HT32 STL 以实现应用和 IEC 60730-1 Class B 的需求。HT32 STL 主程序（主超级循环以低优先级执行的其它测试项）所需的总花费时间也受到应用的影响，例如用户接口设计，因为它通常由主超级循环调用。此外，还应注意完成 HT32 STL 测试的每个测试项所需的总时间，因为它会影响 HT32 STL 检测到错误的响应时间。IEC 60730-1 Class B 定义了各类产品/应用中每个安全相关功能的合理响应时间。

HT32 STL 集成了使用 GPIO 翻转的基本时间测量功能。可以使用逻辑分析器之类的工具来确认每个 HT32 STL 测试项的花费时间。下面的内容显示了 HT32 STL 配置文件（“ht32_stl_config.h”）中集成的基本时间测量的相关配置。IO3 仅用于 HT32 STL 主程序的测量。IO1 和 IO2 用于多种功能（可设置）的测量。前两个设置需设为“1”，其它设置根据需求进行修改。

```
#define HTCFG_STL_DEBUG_EN                  (1) // Global Debug Control
....
#define HTCFG_STL_DEBUG_EN_IO               (1) // Debug IO Control
....
#define HTCFG_STL_DEBUG_EN_MAIN             (1) // Message & IO3 (StartupTest, Routine)
#define HTCFG_STL_DEBUG_EN_MAIN_HS          (1) // IO2 (TimerHandler_HS)
#define HTCFG_STL_DEBUG_EN_INT              (0) // Message & IO3 (IntMonitorHandler)
#define HTCFG_STL_DEBUG_EN_INVARMEM         (0) // IO1 (Cycle Finished), IO2 (Block Finished)
#define HTCFG_STL_DEBUG_EN_VARMEM           (0) // IO1 (Cycle Finished), IO2 (Block Finished)
…
#define HTCFG_STL_DEBUG_IO1_GPIOX           B
                                            6
```

```
#define HTCFG_STL_DEBUG_IO1_GPION       B
#define HTCFG_STL_DEBUG_IO2_GPIOX       7
#define HTCFG_STL_DEBUG_IO2_GPION       B
#define HTCFG_STL_DEBUG_IO3_GPIOX       8
#define HTCFG_STL_DEBUG_IO3_GPION
```

由于大部分测试是由 MCU 代码执行，所以花费时间随 MCU 内核速度的不同而不同。下表显示了 HT32F50241（20 MHz 不带 Flash 加速器）和 HT32F65240（60 MHz 带 Flash 缓存）花费时间的范例。因为测试环境对花费时间结果有很大的影响，如编译器版本、编译器优化设置和 C 代码/函数实现等，所以本范例测试结果可以作为参考，但不能保证。

测试项		**HT32F50241** (20 MHz 不带 Flash 加速器)	**HT32F65240** (60 MHz 带 Flash 缓存)
启动测试 – 非易失性存储器		**2.9** ms (7,176 字节)	**1.1** ms (7,320 字节)
启动测试 – 易失性存储器 March X，8,192 字节		**2.7** ms	**0.9** ms
启动测试 – 存储器总花费时间：		**5.6** ms	**2.0** ms
自检 – CPU 寄存器		**14.4** µs	**5.7** µs
自检 – 程序计数器		**4.5** µs	**2.4** µs
自检 – 中断		**4.8** µs	**2.3** µs
自检 – 非易失性存储器 1 块，16 字节		**15.7** µs	**7.4** µs
自检 – 数字 I/O (2)		N/A	N/A
自检 – 模拟 I/O		**15.6** µs (3)	**7.3** µs (4)
“HT32_STL_MainRoutine()”总花费时间： (5)		**56** µs	**26** µs
自检 – 非易失性存储器全部完成(5)		**27.2** ms (7,176 字节)	**13** ms (7,320 字节)
自检 – 易失性存储器 March X，正常情况：块大小 = 6，重叠 = 2	1 块 (24 字节/10 ms)	**18.7** µs	**8.7** µs
	全部完成(6)	**0.97** s	**0.97** s
自检 – 易失性存储器 March X，最快情况：块大小 = 2，重叠 = 1	1 块 (8 字节/10 ms)	**6.2** µs	**2.8** µs
	全部完成(6)	**3.85** s	**3.85** s

注：

1. CPU 寄存器启动测试的花费时间测量没有集成到 HT32 STL 中。
2. 取决于所使用的 I/O 数量和 STL 函数，不包括在这个测量中。
3. 取决于以下 ADC 设置并转换一个内部 ADC 通道。
   ADC 时钟 20 MHz / 4 = 5 MHz，
   采样时间 = (1.5 + 2 + 12.5)= 16 T
4. 取决于以下 ADC 设置并转换一个内部 ADC 通道。
   ADC 时钟 60 MHz / 4 = 15 MHz，
   采样时间 = (1.5 + 2 + 12.5)= 16 T
5. 包括函数调用和 I/O 控制的花费时间。结果将根据中断的影响而改变。
6. 结果是基于 1,536 字节的 SRAM 测试。“全部完成”时间可以通过以下公式计算：
   全部完成时间 = (存储器大小 / ((块大小 - 重叠) × 4 字节)) × 块测试时间间隔，
   其中，
   存储器大小 = (HTCFG_STL_VARMEM_RUN_CLASSB_END - HTCFG_STL_VARMEM_RUN_CLASSB_START)
   块大小 = HFCFG_STL_VARMEM_RUN_BLOCK_SIZE
   重叠 = HFCFG_STL_VARMEM_RUN_BLOCK_OVERLAP

块测试时间间隔 = HTCFG_STL_VARMEM_RUN_TEST_COUNT_MS
范例：(1536 / ((6 - 2) × 4)) × 10 ms = 960 ms

HT32 STL 调试

HT32 STL 有一些集成的调试功能，如下所示：

调试消息：使用 printf 重定向到 UxART。
调试 I/O：用于前面讨论的时间测量。
LED 指示灯：指示自检失败，已安全停止。
安全停止 ID：指示安全停止的失败源。

下表显示了 HT32 STL 的调试相关设置。这些设置可以在 HT32 STL 配置文件“ht32_stl_60730b_config.h”中找到。有关每个设置交互的详细描述，请参考“ht32_stl_debug.h”文件。

设置名称	说明
全局调试设置	
HTCFG_STL_DEBUG_EN	全局调试控制 0：除能所有调试功能 1：使能调试功能（根据每个测试项的调试设置）
HTCFG_STL_DEBUG_EN_MSG	调试消息控制 0：除能所有调试消息 1：使能调试消息（根据每个测试项的调试设置）
HTCFG_STL_DEBUG_EN_ERR	错误消息控制 0：除能所有错误消息 1：使能错误消息 注：错误消息是调试消息的一种，意味着它是由“HTCFG_STL_DEBUG_EN_MSG”设置控制的。
HTCFG_STL_DEBUG_EN_IO	调试 I/O 控制 0：除能所有调试 I/O 1：使能调试 I/O（根据每个测试项的调试设置）
使能/除能每个测试项的调试功能	
HTCFG_STL_DEBUG_EN_MAIN	消息 & IO3 (启动测试，程序)
HTCFG_STL_DEBUG_EN_MAIN_HS	IO2 (TimerHandler_HS)
HTCFG_STL_DEBUG_EN_PC	消息
HTCFG_STL_DEBUG_EN_INT	消息 & IO3 (IntMonitorHandler)
HTCFG_STL_DEBUG_EN_CLK	消息
HTCFG_STL_DEBUG_EN_INVARMEM	IO1 (循环完成)，IO2 (块完成)
HTCFG_STL_DEBUG_EN_VARMEM	IO1 (循环完成)，IO2 (块完成)

当 HT32 STL 不能按预期工作时，例如由于未知原因进入安全停止函数，首先检查安全停止 ID，以定位安全停止的失败源。安全停止 ID 是一个 16 位编码的参数，由每个 HT32 STL 测试函数传递给安全停止函数。每个 STL 测试函数都有一个唯一的安全停止 ID，安全停止 ID 的格式如下：

0xMMNN

其中“MM”为 HT32 STL 测试项的 ID，“NN”为失败原因的子 ID。实际的 ID 值可以在“ht32_stl_debug.h”文件中找到。例如，0x4102 表示失败源在非易失性存储器测试中，以及失败原因是启动测试。

此外，全局变量冗余的验证函数也会调用安全停止函数，但是它传递给安全停止函数的是存储器失败地址（SRAM 地址），而不是安全停止 ID。对于 Cortex®-Mx 预定义的存储器映射，存储器地址必须是 0x2xxxxxxx。ID 值的第 29 位可用于区分安全停止 ID 和存储器失败地址。

当前的安全停止 ID 可以通过调试错误消息或 Keil uVision 调试模式获得。安全停止 ID 的错误消息如下图所示：

```
STL Main Startup Test ....

HT32_STL_SafeStopCritical(), DID = 0x00004102
```

下图显示了通过 Keil uVision 调试模式获取安全停止 ID 的方法。进入 Keil uVision 调试模式，并在安全停止函数“HT32_STL_SafeStopCritical()”（在”ht32_stl_main.c”中）的开始处设置一个断点，之后可以在寄存器窗口的 R0 中找到安全停止 ID。

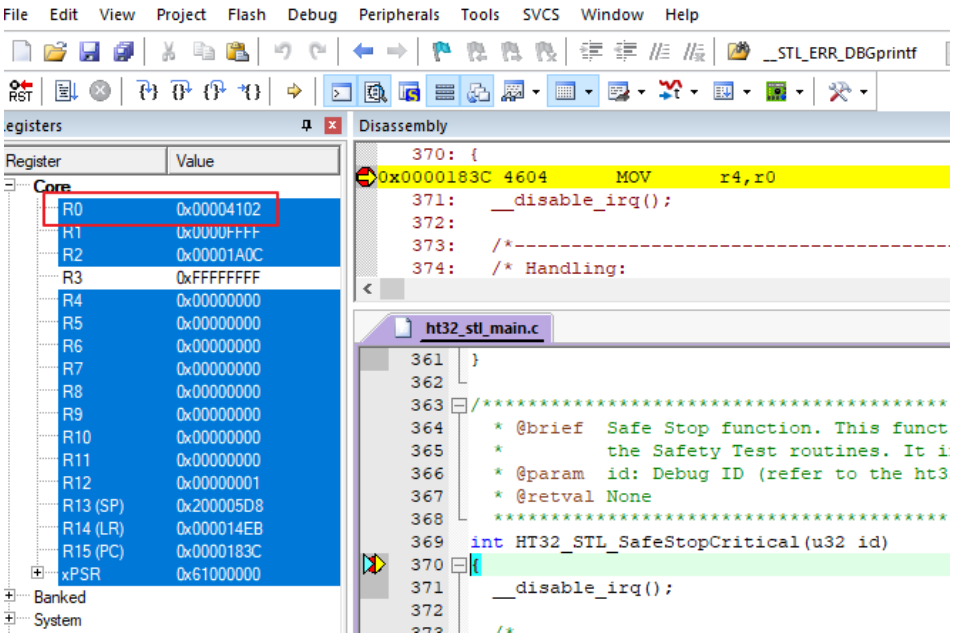


图 20. 使用 Keil uVision 获取安全停止 ID

获取安全停止的失败源之后，可以通过设置“HTCFG_STL_DEBUG_EN”、“HTCFG_STL_DEBUG_EN_MSG”和“HTCFG_STL_DEBUG_EN_XXX”（XXX 表示 HT32 STL 的测试项名称）为 1 来使能更多调试消息。在此之后，可以观察每个 HT32 STL 测试项的调试消息来确定故障的根本原因。

结论

本应用笔记描述了 Holtek HT32 STL 的功能、环境、系统架构和使用说明，并演示了如何将 HT32 STL 集成到用户的固件中。使用 HT32 STL，用户可免去 UL/IEC 60730-1 Class B 规范内要求的元件自检程序开发时间，大幅缩短软件认证周期，节省可观的开发与认证成本。

参考资料

“IEC 60730-1：家用和类似用途的自动电气控制器”，国际电工委员会，3.2 版，2007-03。

HT32 系列 Datasheet 和用户手册。

版本及修改信息

日期	作者	发行	修订信息
2022.06.02	刘威廷	V1.10	修改下载路径
2020.02.27	刘威廷	V1.00	第一版

免责声明

本网页所载的所有数据、商标、图片、链接及其他数据等（以下简称「数据」），只供参考之用，合泰半导体 (中国) 有限公司及其关联企业（以下简称「本公司」）将会随时更改数据，并由本公司决定而不作另行通知。虽然本公司已尽力确保本网页的数据准确性，但本公司并不保证该等数据均为准确无误。本公司不会对任何错误或遗漏承担责任。

本公司不会对任何人士使用本网页而引致任何损害（包括但不限于计算机病毒、系统故障、数据损失）承担任何赔偿。本网页可能会连结至其他机构所提供的网页，但这些网页并不是由本公司所控制。本公司不对这些网页所显示的内容作出任何保证或承担任何责任。

责任限制

在任何情况下，本公司并不须就任何人由于直接或间接进入或使用本网站，并就此内容上或任何产品、信息或服务，而招致的任何损失或损害负任何责任。

管辖法律

以本公司所在地法律为准据法，并以本公司所在地法院为第一审管辖法院。

免责声明更新

本公司保留随时更新本免责声明的权利，任何更改于本网站发布时，立即生效。