Application Note

Safety Startup

Introduction

Safety plays an increasingly important role in the field of electronic applications. In electronic design, the level of component safety requirements is constantly increasing, and electronic device manufacturers are incorporating many new technology solutions into new component designs. Software technologies are constantly emerging to improve safety. Standards for hardware and software safety requirements are also under continuous development.

This document describes how the project in N32G430 MCU to perform the requirements of IEC60730 software safety related operations, as well as related application code content.

This document applies to the N32G430 series products of NSING Technologies.

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road, Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg

Content

1. This Section Describes the IEC60730 Class B Software Standard

To ensure the safety of electrical appliances, risk control measures during software operation need to be evaluated.

IEC60730, issued by the International Electrotechnical Commission, introduces the requirements for the evaluation of software for household appliances. In Appendix H(H.2.21), software is classified as follows:

Class A software: the software only implements the functions of the product and does not involve the safety control of the product. For example, software for room thermostats, lighting controls, etc.

Class B software: the software designed to prevent unsafe operation of electronic devices. For example, the washing machine software with automatic door lock control, the induction cooker software with overheating control, etc.

Class C software: the software designed to avoid certain specific hazards. For example, automatic burner control and thermal cut-off for enclosed water heater (mainly for device that may cause explosions), etc.

The specific evaluation requirements of class B software include components to be tested and related faults and test schemes, which are sorted out in the following table (refer to IEC60730 Table H.11.12.7):

Table 1-1 The Specific Evaluation Requirements of Class B Software

Components to be Detected		Fault/Error	Fault Classification	Nsing with Library	Test Solution Overview
1.CPU	1.1 Register	Hysteresis (Stuck at)	MCU related	Y	Write relevant registers and check
	1.3 Program counter	Hysteresis (Stuck at)	MCU related	Y	When the PC crash, start the watchdog reset
2.Interruption		No interrupts or interrupts too frequently	Application related	N	Count the number of interrupts
3. The clock		Incorrect frequency	MCU related	Y	Use HSI to measure HSE clock frequency
4. Memory	4.1 Non-volatile memory	All single bit errors	MCU related	Y	Flash CRC integrity check
	4.2 Volatile memory	DC fault	MCU related	Y	1. SRAM March C test 2. Stack overflow detection
	4.3 Addressing (related to non-volatile and volatile memory)	Hysteresis (Stuck at)	MCU related	Y	Flash/SRAM tests are included

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road, Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg

5. Internal data path	5.1 Data	Hysteresis (Stuck at)	MCU related	N	Only for MCU using external memory, monolithic MCU is not required
	5.2 Addressing	Incorrect address	MCU related	N	
External communication	6.1 Data	The Hamming distance is 3	Application related	N	Add verification in data transfer
	6.2 Addressing	Incorrect address	Application related	N	
	6.3 Timing	Incorrect timing	Application related	N	Count the number of communication events
7. Input and output	7.1 Digital I/O	Error defined in H27	Application related	N	None
	7.2 Analog input and output	Error defined in H27	Application related	N	None

2. Test Point Process Description

Class B software package program detection content is divided into two main parts: self-check at startup and periodic self-check at runtime. Self-check at startup includes:

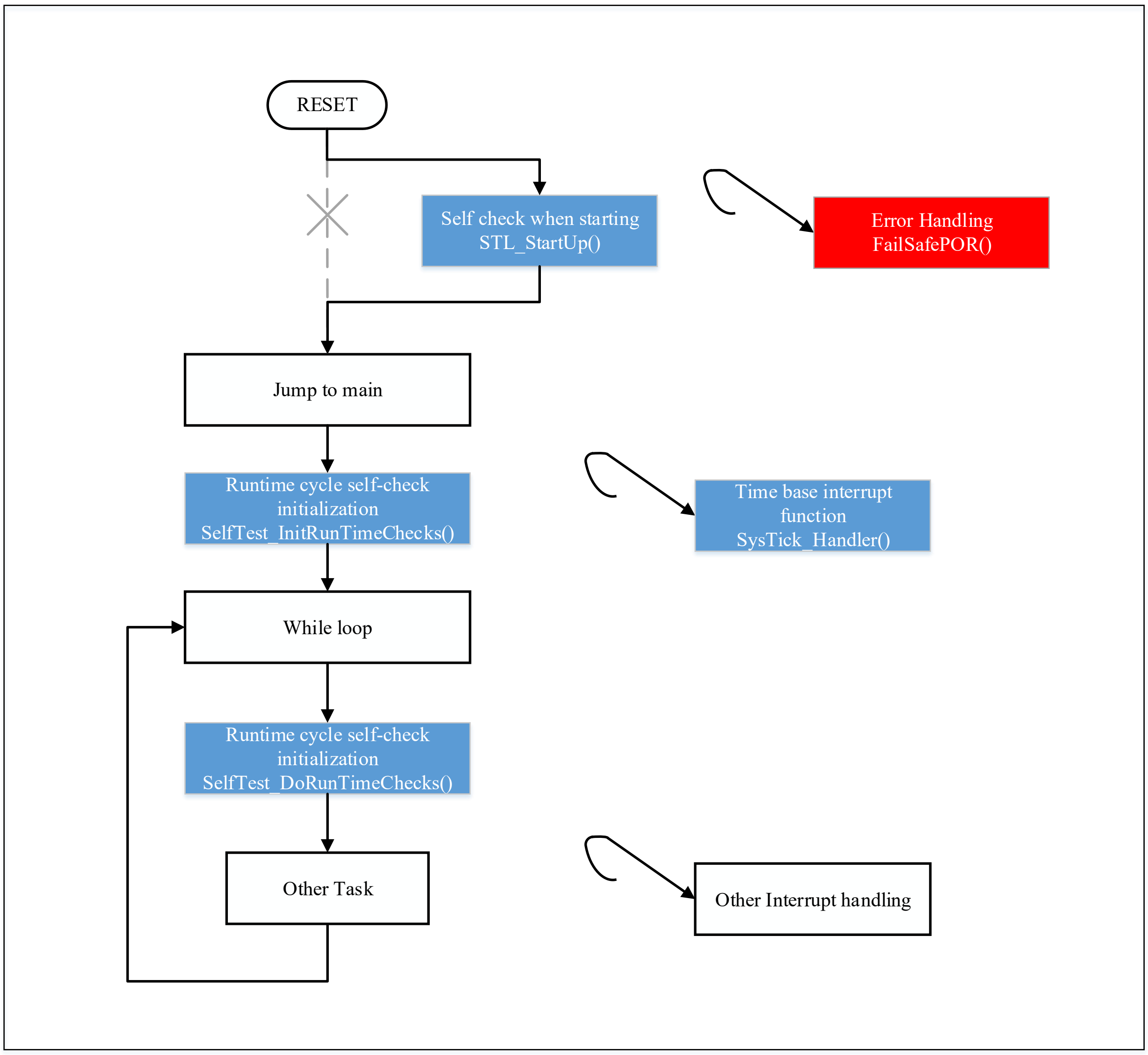
- CPU self-check
- Watchdog self-check
- Flash integrity self-check
- RAM function self-check
- System clock self-check
- Control flow self-check

Periodic self-check at runtime:

- Local CPU core register self-check
- Stack boundary overflow self-check
- System clock running self-check
- Flash CRC segmentation self-check
- Watchdog self-check
- Partial RAM self-check (in interrupt service routines)

The overall flow diagram is as follows:

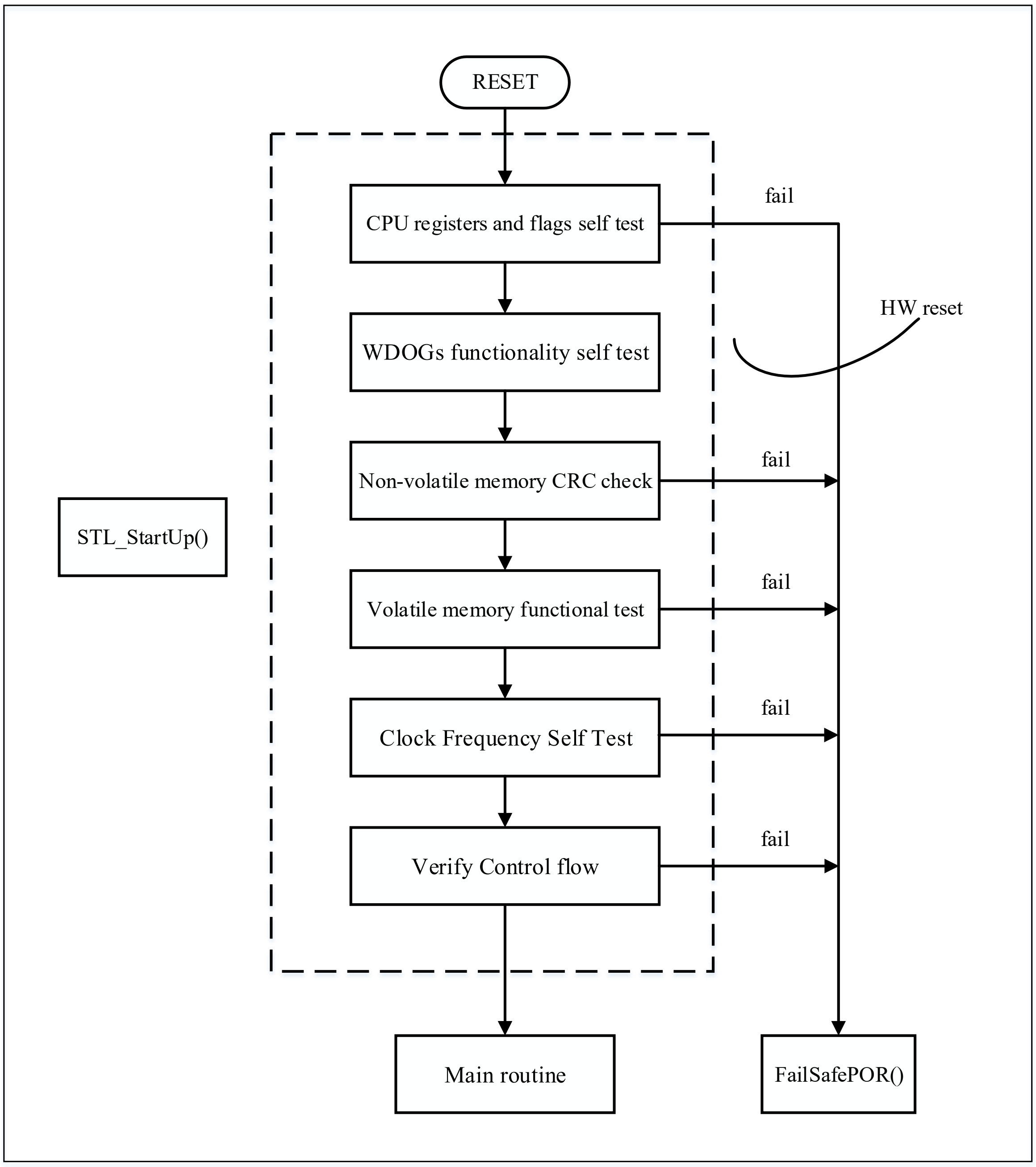
Figure 2-1 Class B Software Self-Check Flow Diagram



2.1 Check Flow at Startup

Before the chip enters **main** function from startup, the startup self-check is performed first, and the startup file is modified to execute this part of the code. After the self-check process is completed, the **__iar_program_start** function is called to jump back to **main** function.
The following is a flow diagram for performing a startup self-check:

Figure 2-2 Class B Software Self-Check Flow Diagram


RESET
CPU registers and flags self test
WDOGs functionality self test
Non-volatile memory CRC check
Volatile memory functional test
Clock Frequency Self Test
Verify Control flow
STL_StartUp()
fail
fail
fail
fail
fail
HW reset
Main routine
FailSafePOR()

2.1.1 CPU Startup Detection

CPU self-check mainly checks whether the core flags, registers and so on are correct. If an error occurs, the fault-safe handling function **FailSafePOR ()** is called.
CPU self-check is performed both at startup and runtime. At startup, a functional test of the flags in R0~R12, PSP, MSP register and Z(zero), N(negative), C(carry), V(overflow) is conducted once as part of self-check process; at runtime, periodic self-check only detect registers R1~R12.

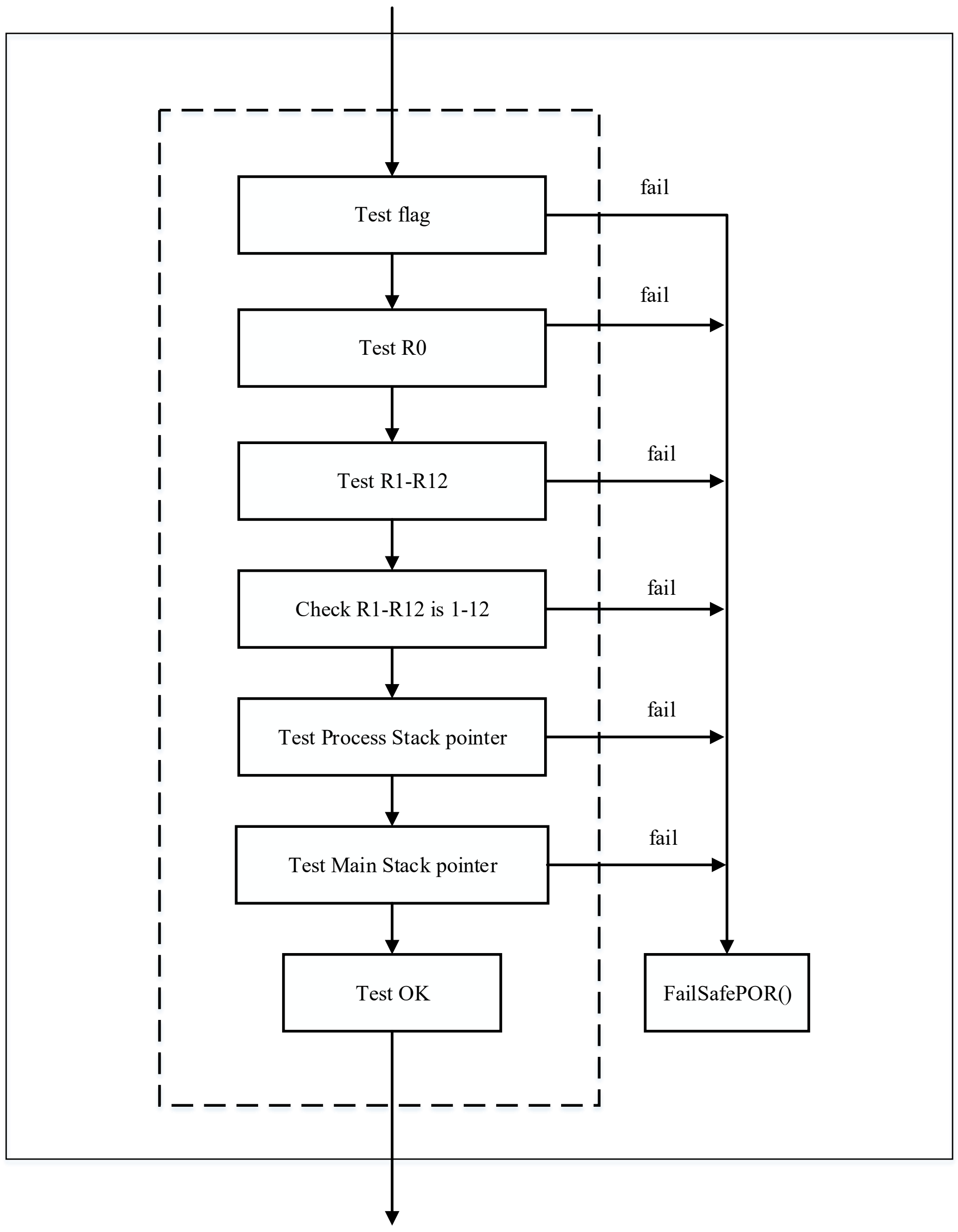
The specific implementation method for register self-check is as follows: write 0xAAAAAAAA and 0x55555555 to the registers respectively, and then compare the read value if they match the written value. Write 1 after R1 is tested, write 2 after R2 is tested, and so on.

The specific implementation method of flag self-check is as follows: set the flag respectively. If

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road, Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg

any flag is checked incorrectly, proceed the fault handling function. The self-check flow diagram is as follows:

Figure 2-3 CPU Startup Self-check Flow Diagram



2.1.2 Watchdog Startup Self-check

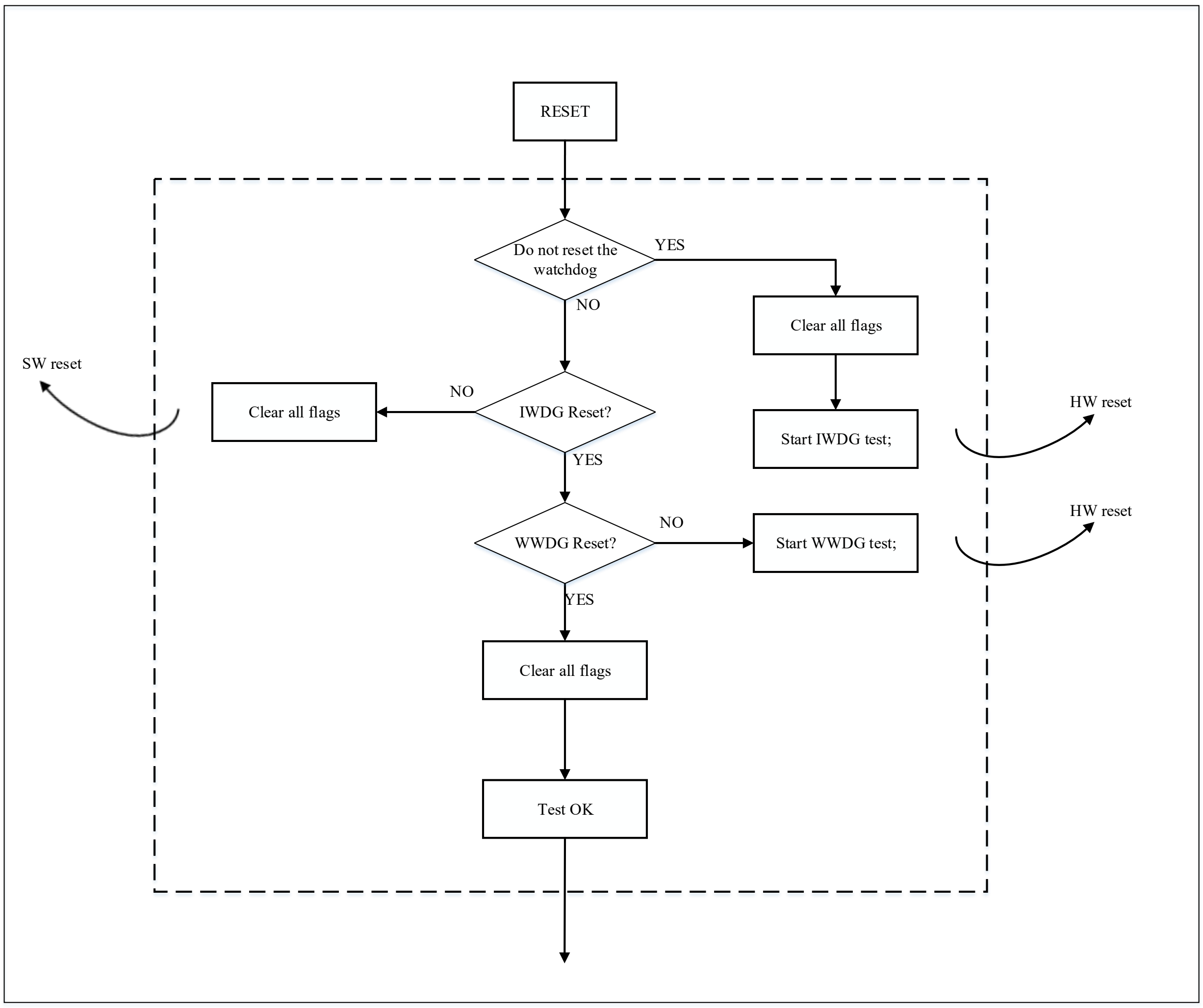
Test to verify that independent watchdog and window watchdog can be reset correctly, ensuring that it can be reset in time to prevent jam during program execution.

After the initial reset, clear all reset status register flags, start the IWDG test to reset the chip, and judge whether it is the IWDG reset flag bit; if it is set, start the WWDG test to reset the chip; if the WWDG reset flag bit is set, the watchdog test passes, clear all flags.

The flow diagram is as follows:

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road, Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg

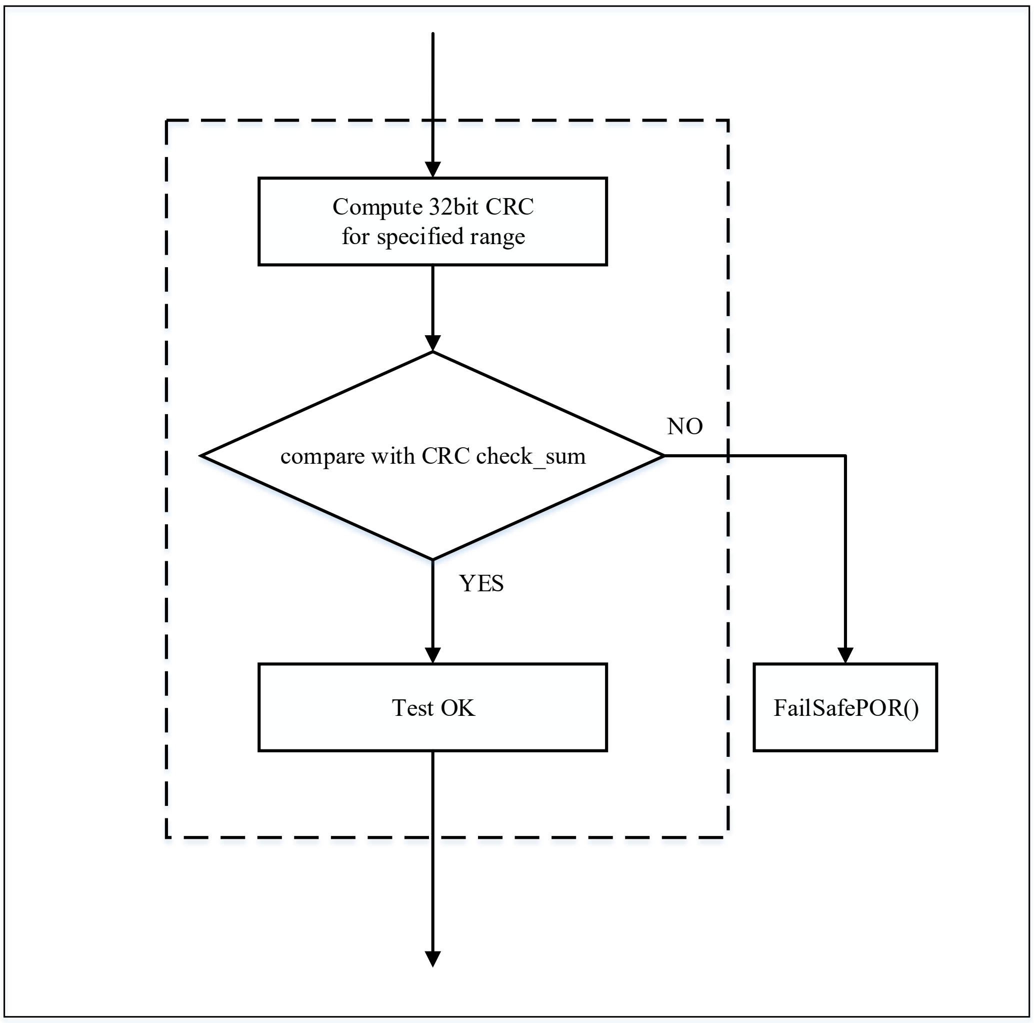
Figure 2-4 Watchdog Startup Self-check Flow Diagram



2.1.3 FLASH Startup Self-check

Flash self-check is a program that calculates Flash data with CRC algorithm and compares the result value with the CRC value calculated during compilation and stored in the specified location of Flash to confirm the integrity of Flash.
The flow diagram is as follows:

Figure 2-5 Flash Startup Self-check Flow Diagram

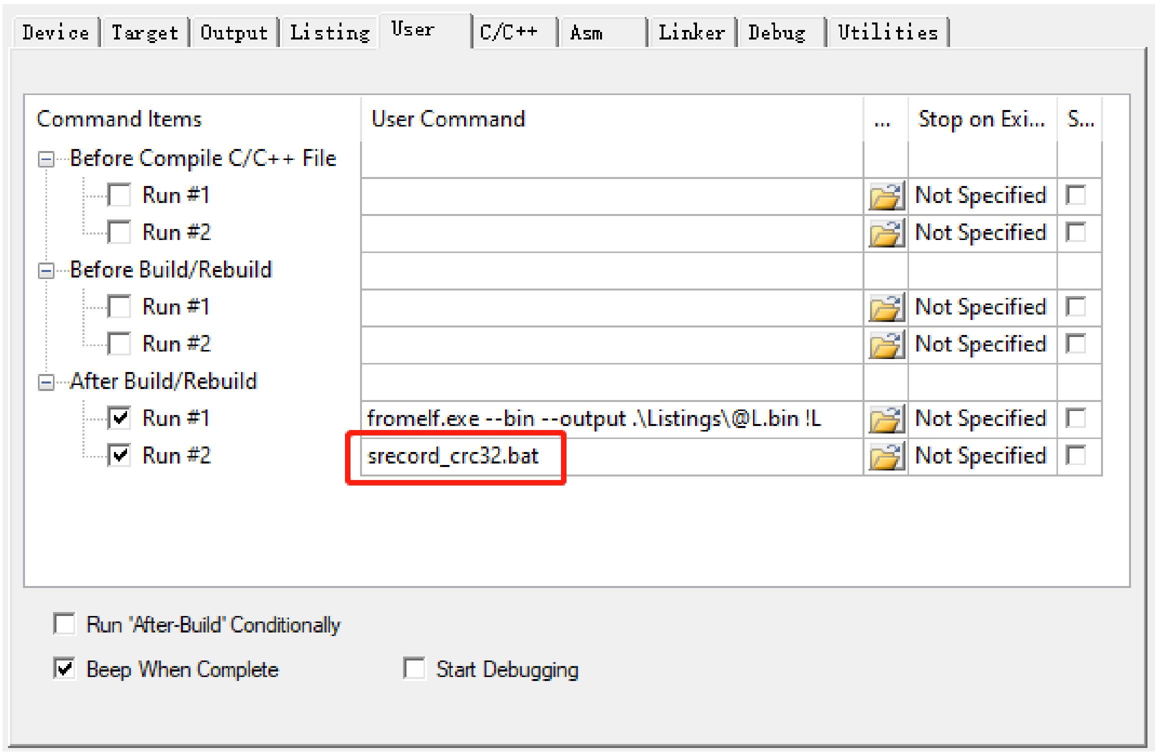

Compute 32bit CRC for specified range
compare with CRC check_sum
NO
YES
Test OK
FailSafePOR()

Flash range for CRC calculation is configured according to the actual situation of the entire program:

The configuration of Keil is more complicated. ARM officially recommends using the third-party software SRecord for ROM Self-Test in MDK-ARM.

According to the project configuration, after the compilation is completed, the script file srecord_crc32.bat will be called. Through the srec_cat.exe software, the data in the N32G430_SelfTest.hex file generated by Keil compilation will be calculated to generate CRC verification result. This result is added to the specified location to obtain a new N32G430_SelfTest_CRC.hex file:

Figure 2-6 Flash Calculation Range Configuration at Linker



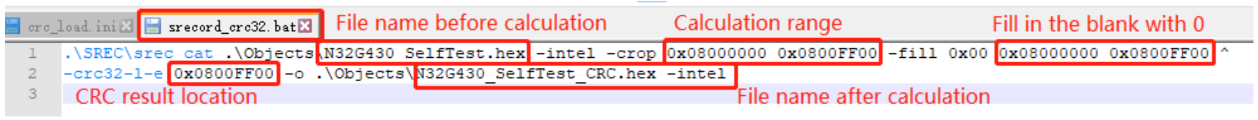
Command Items	User Command	...	Stop on Exi...	S...
Before Compile C/C++ File				
☐ Run #1			Not Specified	☐
☐ Run #2			Not Specified	☐
Before Build/Rebuild				
☐ Run #1			Not Specified	☐
☐ Run #2			Not Specified	☐
After Build/Rebuild				
☑ Run #1	fromelf.exe --bin --output .\Listings\@L.bin !L		Not Specified	☐
☑ Run #2	srecord_crc32.bat		Not Specified	☐

☐ Run 'After-Build' Conditionally

☑ Beep When Complete ☐ Start Debugging

Open the .bat file with Notepad or other tools, and modify the following according to the actual application:

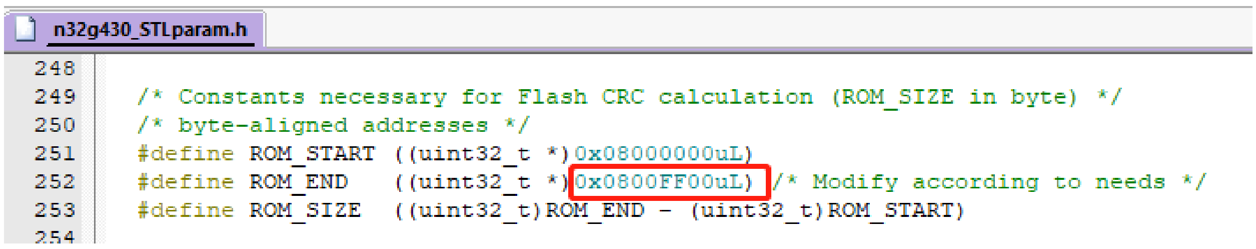
Figure 2-7 Configuration of srecord_crc32.bat



```
.\SREC\srec_cat .\Objects\N32G430_SelfTest.hex -intel -crop 0x08000000 0x0800FF00 -fill 0x00 0x08000000 0x0800FF00 ^
-crc32-l-e 0x0800FF00 -o .\Objects\N32G430_SelfTest_CRC.hex -intel
```

Range of calculating CRC in the program is configured in the n32g430_STLparam.h file, which can be modified according to the requirements to align with the above configuration:

Figure 2-8 Configuration of Range of CRC Calculation



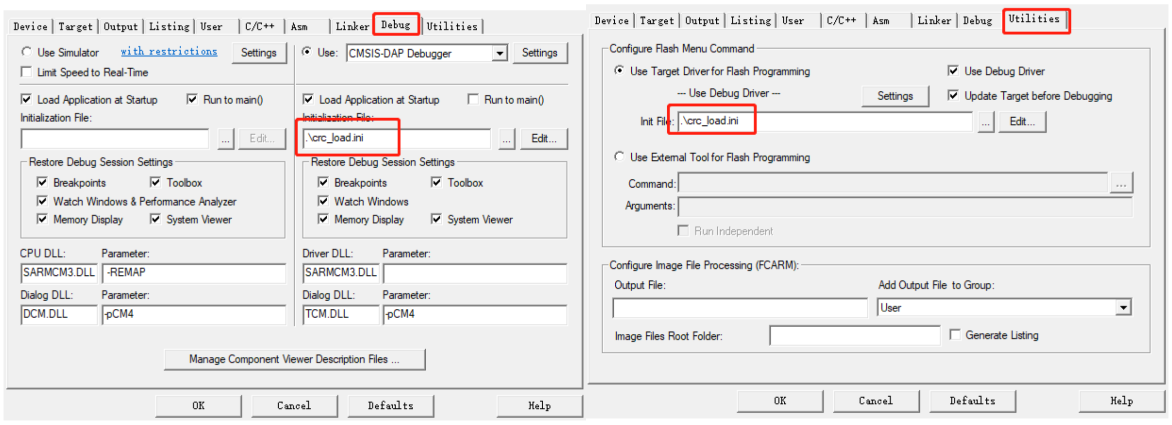
```

/* Constants necessary for Flash CRC calculation (ROM_SIZE in byte) */
/* byte-aligned addresses */
#define ROM_START ((uint32_t *)0x08000000uL)
#define ROM_END   ((uint32_t *)0x0800FF00uL) /* Modify according to needs */
#define ROM_SIZE  ((uint32_t)ROM_END - (uint32_t)ROM_START)
```

Therefore, whether it is downloading or debugging, the final generated N32G430_SelfTest_CRC.hex file needs to be used, so the .ini file needs to be added to the Keil configuration option to download the new .hex file. The configuration is as follows:

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road, Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg

Figure 2-9 .ini File is Added to Keil Configuration



Debug tab:

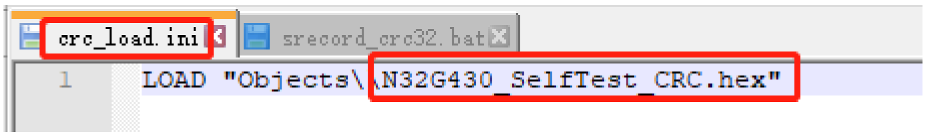
- ○ Use Simulator with restrictions
- ☐ Limit Speed to Real-Time
- ☑ Load Application at Startup ☑ Run to main()
- Initialization File:
- Restore Debug Session Settings: ☑ Breakpoints ☑ Toolbox ☑ Watch Windows & Performance Analyzer ☑ Memory Display ☑ System Viewer
- CPU DLL: SARMCM3.DLL Parameter: -REMAP
- Dialog DLL: DCM.DLL Parameter: -pCM4
- ◉ Use: CMSIS-DAP Debugger
- ☑ Load Application at Startup ☐ Run to main()
- Initialization File: .\crc_load.ini
- Restore Debug Session Settings: ☑ Breakpoints ☑ Toolbox ☑ Watch Windows ☑ Memory Display ☑ System Viewer
- Driver DLL: SARMCM3.DLL Parameter:
- Dialog DLL: TCM.DLL Parameter: -pCM4

Utilities tab:

- Configure Flash Menu Command
- ◉ Use Target Driver for Flash Programming ☑ Use Debug Driver
- -- Use Debug Driver -- ☑ Update Target before Debugging
- Init File: .\crc_load.ini
- ○ Use External Tool for Flash Programming
- Command:
- Arguments:
- ☐ Run Independent
- Configure Image File Processing (FCARM):
- Output File: Add Output File to Group: User
- Image Files Root Folder: ☐ Generate Listing

It should be noted that the .ini file should be configured with the file name of the actual application requirements that need to be modified

Figure 2-10 Configuration of crc_load.ini



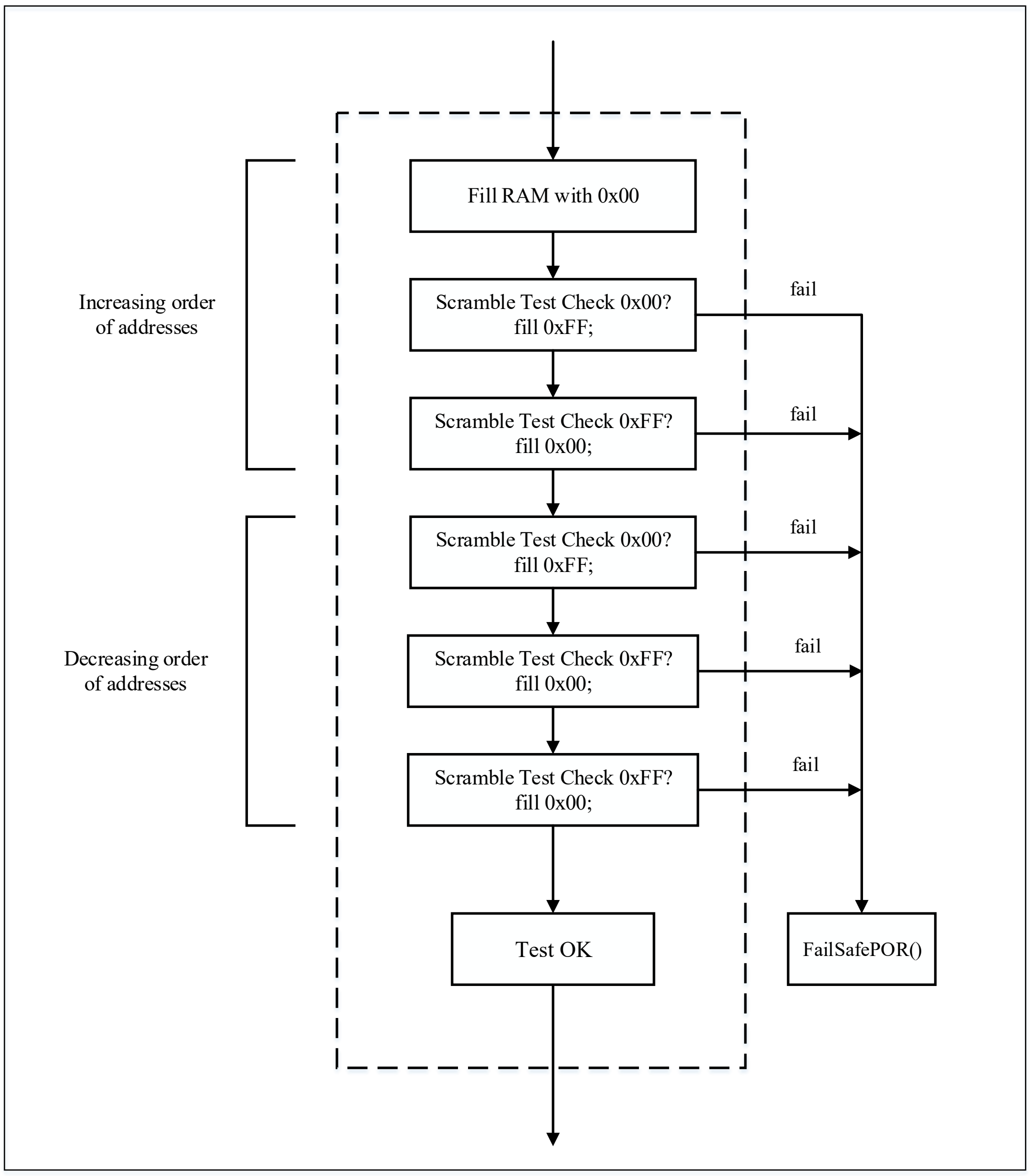
```
LOAD "Objects\N32G430_SelfTest_CRC.hex"
```

2.1.4 RAM Startup Self-test

SRAM self-check detects errors not only in the data region, but also in its internal address and data path.
SRAM self-check uses the March-C algorithm, which is an algorithm used for SRAM testing of embedded chips and is a part of safety certification. All ranges of SRAM are detected at startup. Firstly, the entire SRAM is cleared, and then each bit is set to 1 sequentially. After setting each bit, it is tested whether the bit is indeed 1. If it is, the process continue; if not, an error is reported. After all bits are set, each bit is cleared 0 sequentially. After clearing each bit, it is tested whether the bit is cleared to 0. If it is, it is considered correct; otherwise, an error is reported. This process continues until the entire RAM space is test.
The test is divided into 6 loops, and the entire RAM is checked and filled word by word alternately with the values 0x00 and 0xFF. The first 3 loops are executed in increasing address order, and the last 3 loops are executed in decreasing address order.
The entire RAM self-test algorithm process is shown in the figure below:

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road, Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg

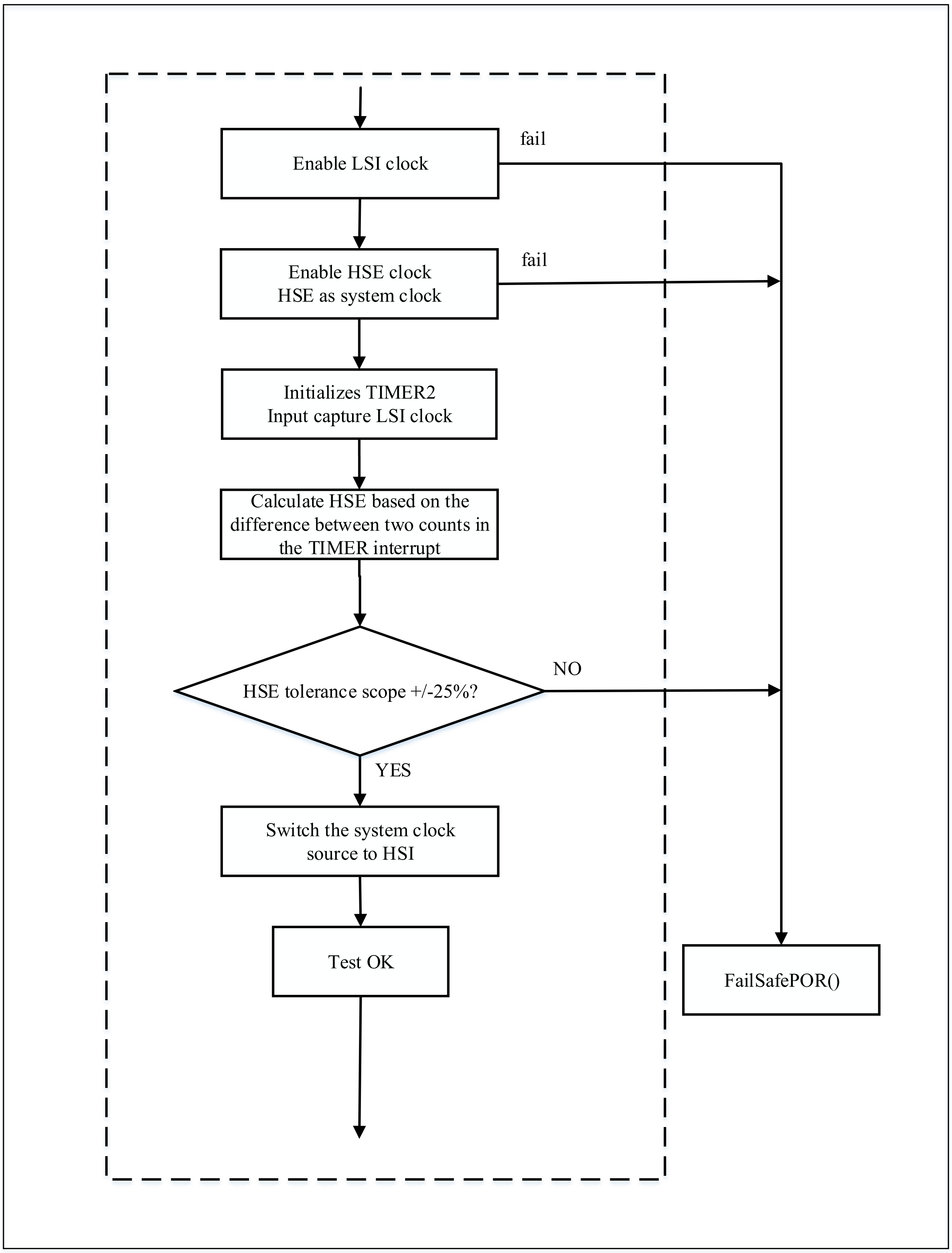
Figure 2-11 RAM Startup Self-check Flow Diagram


Fill RAM with 0x00
Scramble Test Check 0x00? fill 0xFF;
Scramble Test Check 0xFF? fill 0x00;
Scramble Test Check 0x00? fill 0xFF;
Scramble Test Check 0xFF? fill 0x00;
Scramble Test Check 0xFF? fill 0x00;
Test OK
FailSafePOR()
fail
Increasing order of addresses
Decreasing order of addresses

2.1.5 Clock Startup Self-test

The test process is as follows:
1. Start the low-speed internal clock (LSI) source.
2. To measure HSE, selects HSE in the macro definition, then starts the high-speed external clock (HSE) source, and configures it as the system clock; otherwise selects HSI in the macro definition, and configures the system clock to select PLL (the source is HSI).
3. Initialize TIMER2 for input capture with LSI clock; in the interrupt, determine the value obtained by the timer counter two consecutive times are different, so that the ratio between the LSI and HSE frequencies can be obtained.
4. Calculate the HSE frequency and compare the frequency value to the expected range value: if it exceeds +/- 25%, the test fails. Switch the system clock source to HSI after the test. The expected range value can be adjusted by the user according to the actual application. The macros are defined as HSE_LimitHigh() and HSE_LimitLow().

Figure 2-12 Clock Startup Self-check Flow Diagram


Enable LSI clock
fail
Enable HSE clock
HSE as system clock
fail
Initializes TIMER2
Input capture LSI clock
Calculate HSE based on the difference between two counts in the TIMER interrupt
HSE tolerance scope +/-25%?
NO
YES
Switch the system clock source to HSI
Test OK
FailSafePOR()

2.1.6 Control Flow Startup Self-test

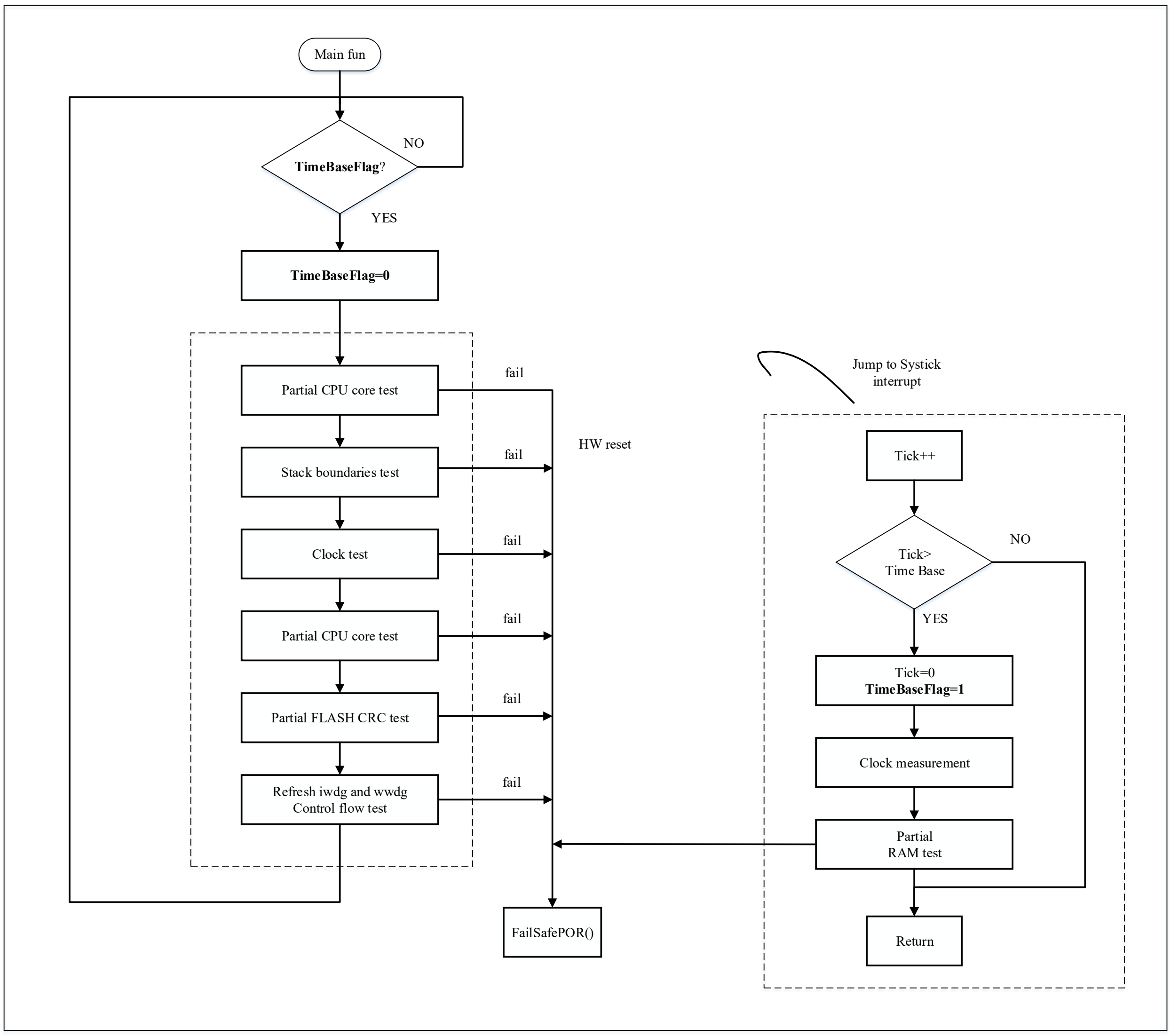
The self-check part of the startup ends with the control flow self-check pointer program.

Initialize the variables CtrlFlowCnt to 0, CtrlFlowCntInv to 0xFFFFFFFF. In each test step, CtrlFlowCnt adds a fixed value, CtrlFlowCntInv subtracting the same fixed value. At the end of the start self-check, judge whether the sum of the two values is still 0xFFFFFFFF.

2.2 Run time inspection process

If the startup self-check passes successfully, the run-time periodic self-check must be initialized before entering the main loop.
The runtime self-check is performed periodically based on SysTick.
The run-time periodic self-check process is as follows:

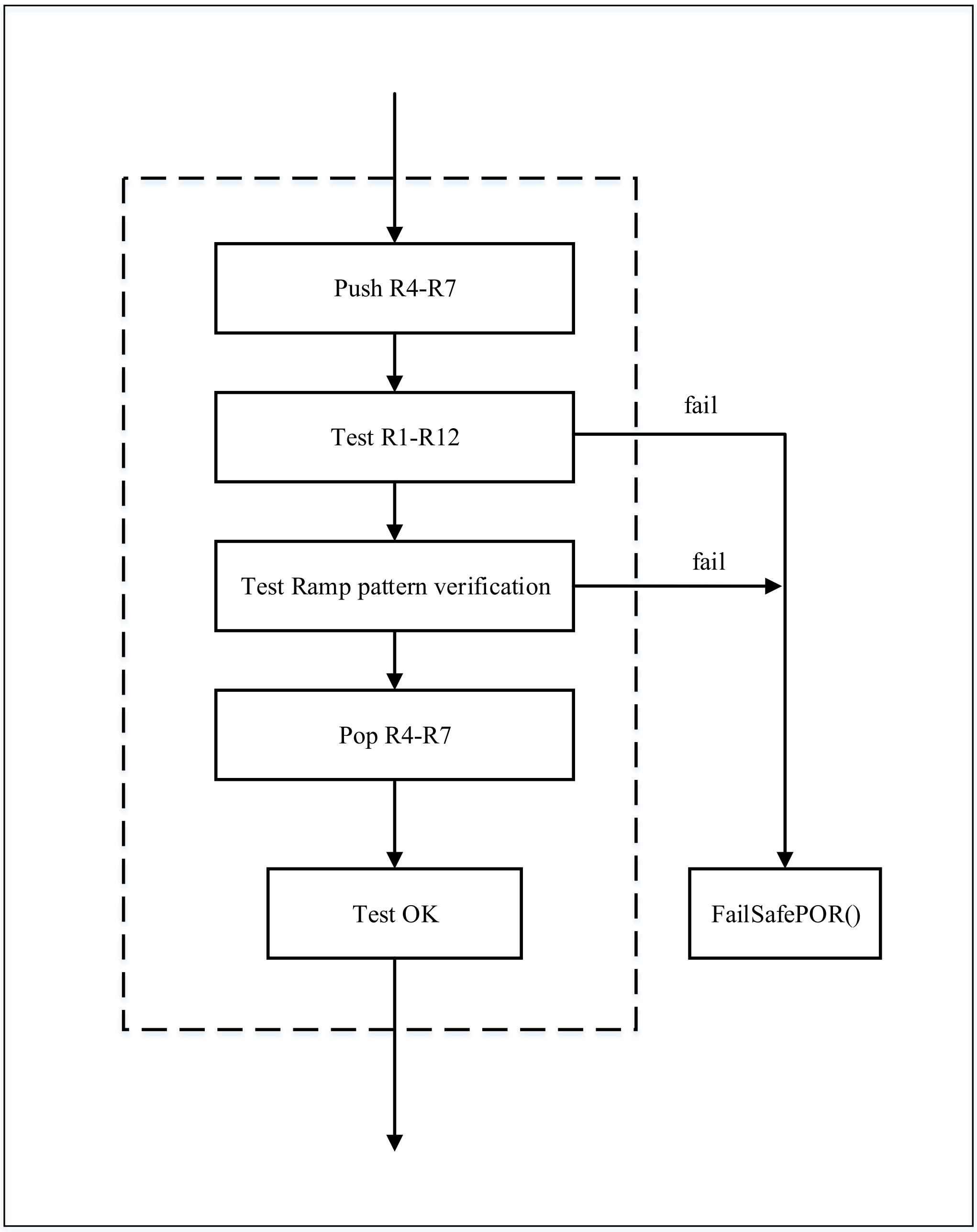
Figure 2-13 Runtime Self-check Flow Diagram



2.2.1 CPU Runtime Self-check

The CPU runtime periodic self-check is like the self-check at startup, except that the core flags and stack pointers are not detected.

Figure 2-14 CPU Runtime Self-check Flow Diagram



2.2.2 Stack Boundary Runtime Overflow Self-check

This test detects stack overflow by determining the data integrity of pattern array in the boundary detection area. If the original pattern data is corrupted, the test fails and a fail-tolerant program is invoked.

The lower address closely following the stack area is defined as the stack boundary detection area. This area can be configured differently depending on the device. The user must define enough areas for the stack and ensure that pattern is placed correctly.

Figure 2-15 Stack Area Diagram

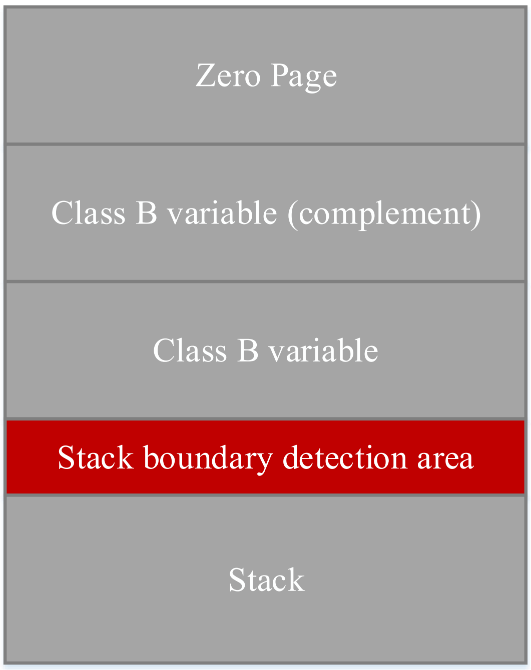
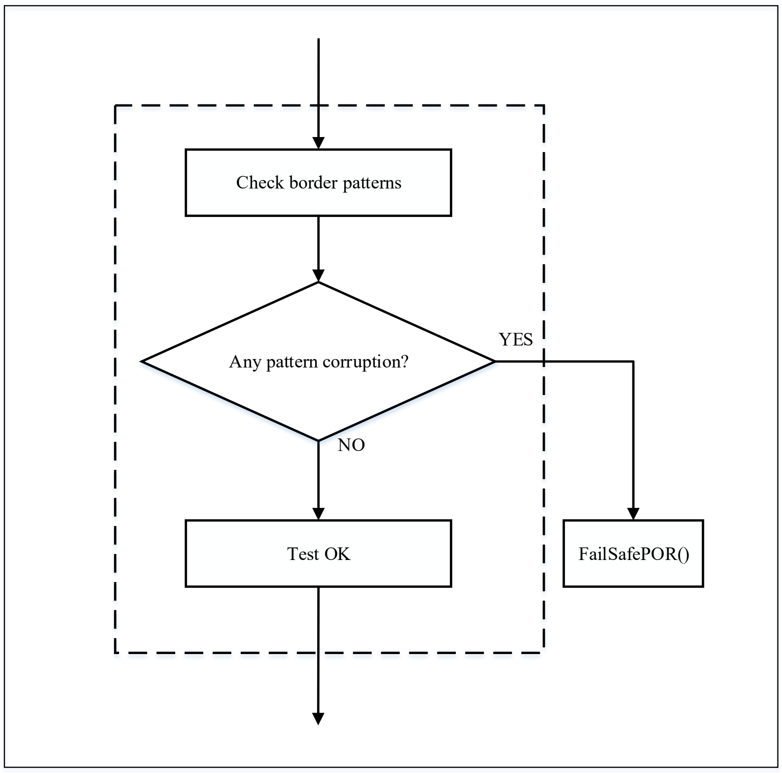


Figure 2-16 Stack Boundary Overflow Runtime Self-check Flow Diagram

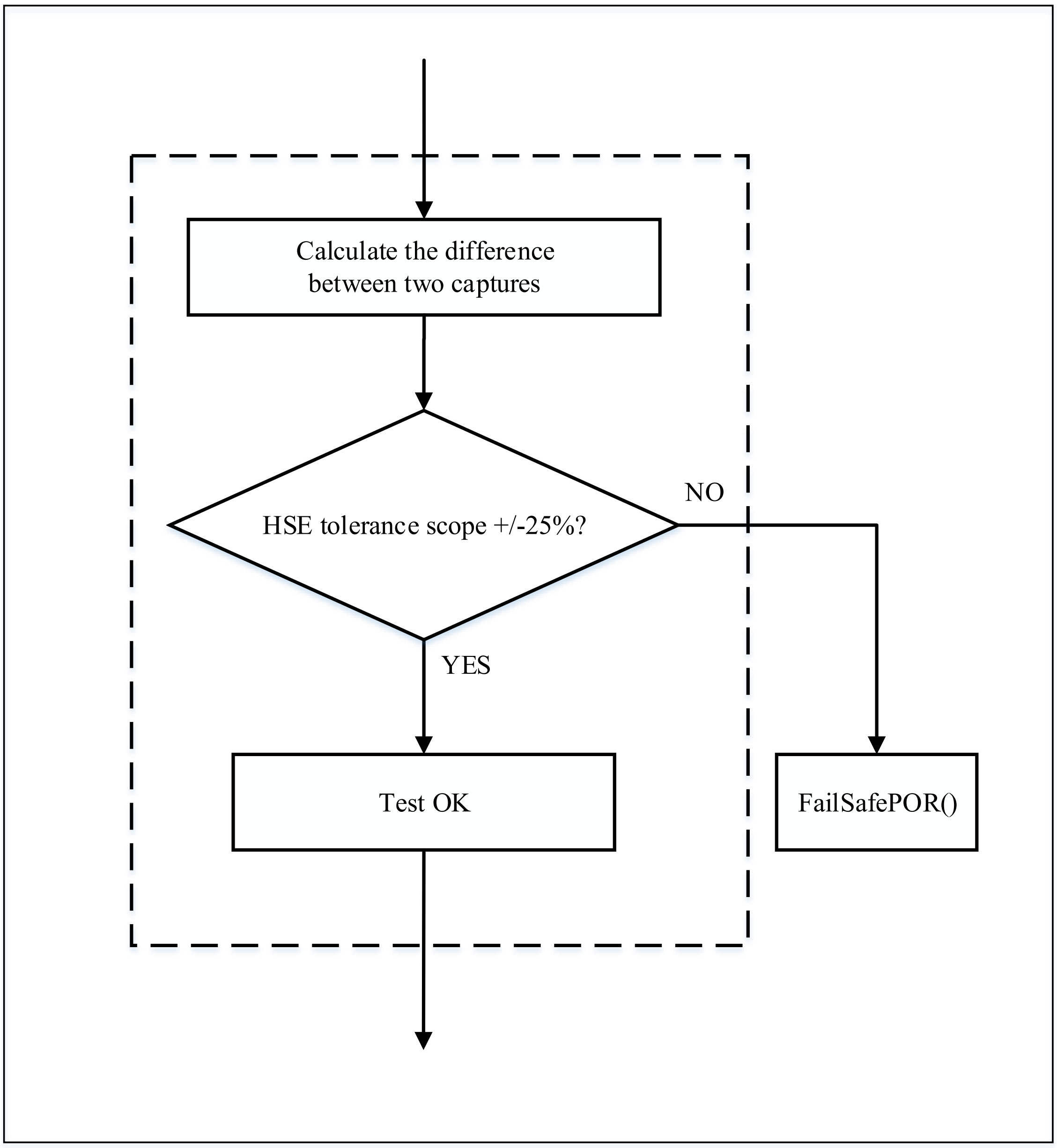


2.2.3 System Clock Runtime Self-check

The self-check of the system clock at runtime is similar to the detection of the clock at startup. The HSE frequency is calculated from the difference between the two captures. The process is as follows:

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road, Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg

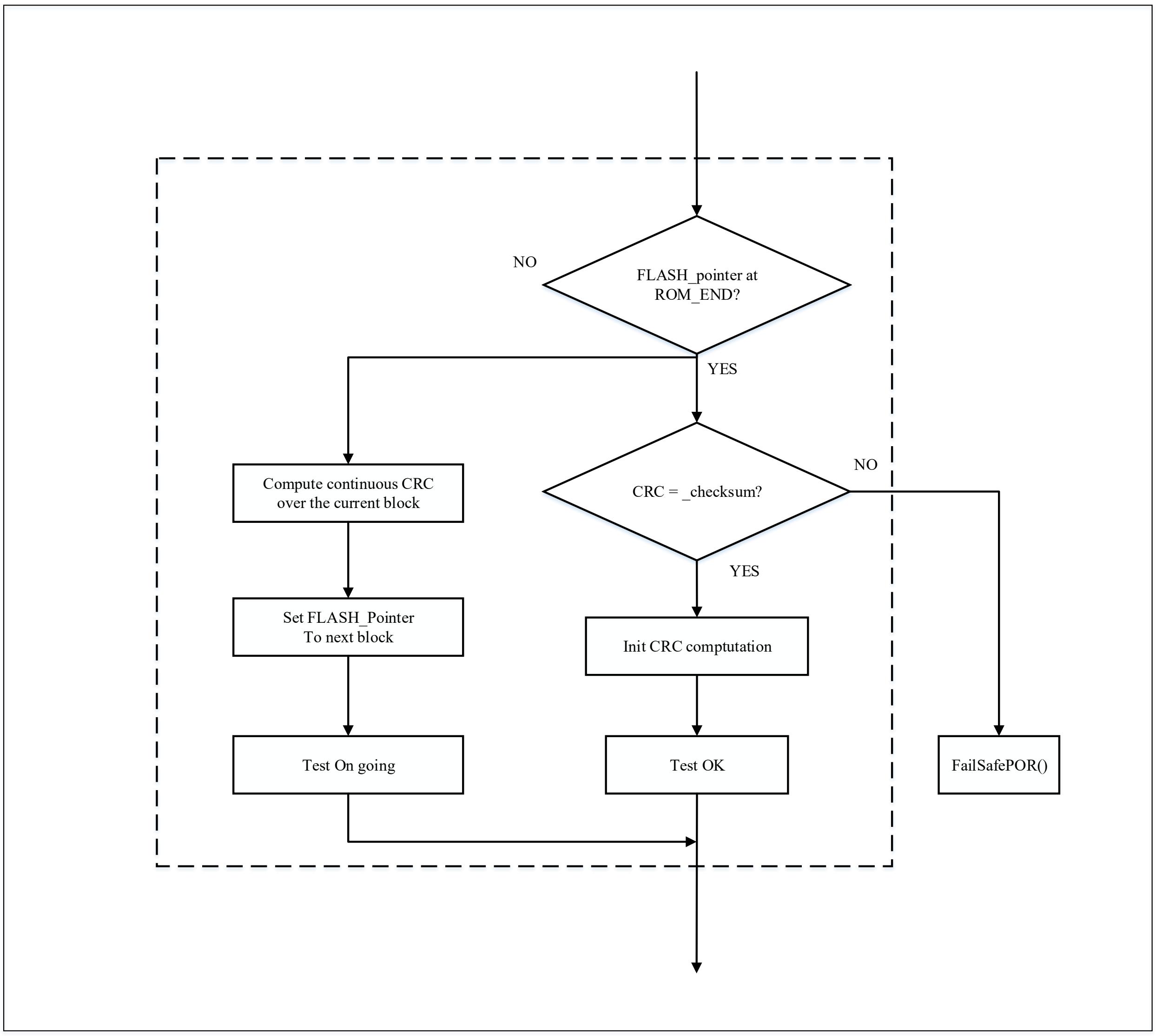
Figure 2-17 System Clock Runtime Self-check Flow Diagram


Calculate the difference between two captures
HSE tolerance scope +/-25%?
NO
YES
Test OK
FailSafePOR()

2.2.4 FLASH Runtime Self-check

The Flash CRC self-check can be performed during the runtime. Because the self-check time length varies with the check range required, user can configure segmented CRC calculation based on the size of the user application. When the CRC values are calculated for the last segment range, the CRC values are compared. If they do match, the test fails.

Figure 2-18 Flash Runtime Self-check Flow Diagram


FLASH_pointer at ROM_END?
NO
YES
Compute continuous CRC over the current block
CRC = _checksum?
NO
YES
Set FLASH_Pointer To next block
Init CRC comptutation
Test On going
Test OK
FailSafePOR()

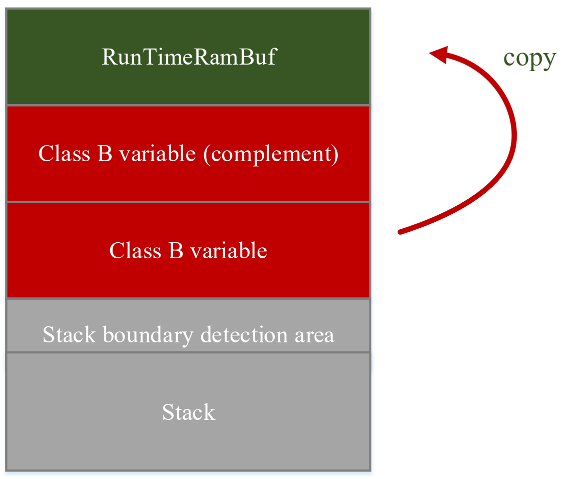
2.2.5 Watchdog Runtime Self-check

During runtime, watchdog needs to be fed periodically to ensure the normal operation of the system. The watchdog feeding part is placed at the end of STL_DoRunTimeChecks().

2.2.6 Local RAM Runtime Self-check

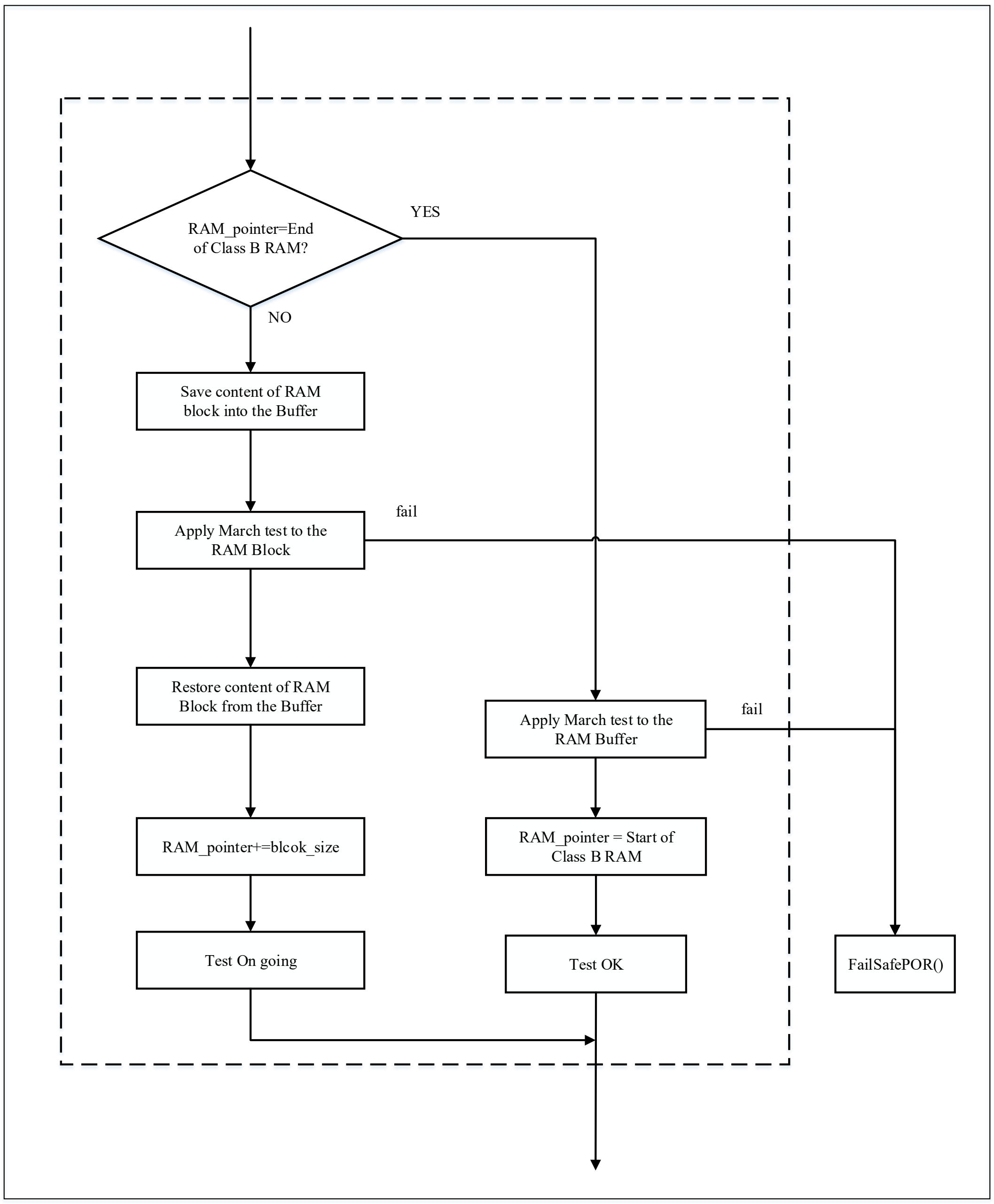
The RAM self-check at run time is performed in the SysTick interrupt function.The test covers only the portion of memory allocated to the class B variables.
The regions allocated to the class B variables are divided into blocks, which each block consisting of 6 bytes. Before the March-C test, save the block data in the RunTimeRamBuf, and then put the RunTimeRamBuf back to the original area of the class B after the test is completed. Until all tests in the class B area are completed.
After the class B region test is completed, March-C tested is performed on the RunTimeRamBuf region. After the test is completed, the pointer is restored to the class B start address for the next test.

Figure 2-19 Local RAM Region Diagram



NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road,
Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg

Figure 2-20 Local RAM Runtime Self-check Flow Diagram


RAM_pointer=End of Class B RAM?
YES
NO
Save content of RAM block into the Buffer
Apply March test to the RAM Block
fail
Restore content of RAM Block from the Buffer
Apply March test to the RAM Buffer
fail
RAM_pointer+=blcok_size
RAM_pointer = Start of Class B RAM
Test On going
Test OK
FailSafePOR()

3. Key Points of Software Library Migration

- Before executing the user program, execute the STL_StartUp function (to start the self-check);
- Set WWDG and IWDG to prevent them from being reset when the program is running properly;
- Set up RAM and Flash self-check range for startup and runtime;
  - The range of CRC checksum, and the location where the checksum is stored in the Flash
  - The range of storage addresses for ClassB variables
  - Location of stack boundary self-check region
- Troubleshoot detected faults.
- Add user-related fault detection content based on specific applications;
- Define the frequency of program runtime self-check according to the specific application;
- After the chip is reset, the STL_StartUp function must be called for startup self-check before initialization.
- Call STL_InitRunTimeChecks() before entering the main loop, and call STL_DoRunTimeChecks() in the main loop;
- Users can release verbose comments to enter diagnostic mode and output text information through the Tx pin (PA9) of USART1.

Set the serial port to 115200Bits/s, no parity, 8-bit data, and 1 stop bit.

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road, Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg

4. Version History

Version	Date	Changes
V1.0	2022.04.07	Initial release

5. Disclaimer

NSING Technologies Pte. Ltd.
Add: NSING, Teletech Park #02-28, 20 Science Park Road, Singapore 117674
Tel: +65 69268090
Email: sales@nsing.com.sg