Application note

N32G033 series 60730 ClassB application note

Introduction

Security plays an increasingly important role in the field of electronic applications. In electronic design, the level of security requirements for components continues to rise, and electronic device manufacturers have incorporated many new technological solutions into new component designs. Software technologies for improving security are constantly emerging. The relevant standards for hardware and software security requirements are also being continuously developed.

NSING provides a safety related self-test library, and this document introduces how the project executes software safety related operations and related application code content required by IEC 60730 in MCU.

NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

Content

Content ... **II**

1. Hardware requirements ... **1**

2. Terminology and Abbreviations ... **3**

2.1 Terms and abbreviations used in this document ... 3

3. Software Package Architecture ... **4**

3.1 Software Architecture Design Description ... 4

3.2 Software Architecture Block Diagram ... 5

4. Software package components ... **6**

4.1 Overview of Software Components ... 6

4.2 Software Component Interface Definition ... 6

5. Software Package Design ... **8**

5.1 Module Detailed Design Description ... 8

5.2 Module Detailed Design Description ... 9

5.2.1 CPU startup detection ... 10

5.2.2 Watchdog startup detection ... 11

5.2.3 FLASH startup detection ... 12

5.2.4 RAM startup detection ... 16

5.2.5 Clock startup detection ... 18

5.2.6 Feature value writing stack boundary ... 19

5.2.7 Detection during control flow startup ... 19

5.3 Run time self-test initialization ... 21

5.4 Run time detection process ... 21

5.4.1 CPU runtime detection ... 22

5.4.2 Stack Boundary Runtime Overflow Detection ... 23

5.4.3 Clock runtime detection ... 24

5.4.4 FLASH runtime detection ... 25

5.4.5 Watchdog runtime refresh ... 26

5.4.6 RAM runtime self-test ... 26

NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

5.4.7 Control flow runtime detection 29

6. Precautions 30

7. Version history 31

8. Notice 32

NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

1. Hardware requirements

To ensure the safety of electrical appliances, it is necessary to evaluate the risk control measures during software operation. The IEC60730 standard issued by the International Electrotechnical Commission introduces the requirements for evaluating household appliance software, and Appendix H (H.2.21) classifies the software:

Class A software: The software only implements the functions of the product and does not involve product security control. For example, software for indoor thermostats and lighting control;

B-class software: The design of the software should prevent unsafe operation of electronic devices. For example, washing machine software with automatic door lock control and induction cooker software with overheating control;

C-class software: The software is designed to avoid certain special hazards. For example, automatic burner control and thermal cut-off of enclosed water heaters (mainly for equipment that can cause explosions).

The specific evaluation requirements for Class B software include the components that need to be tested, related faults, and testing plans. According to the IEC60730 Class-B certification requirements, the public provides an STL library to help users quickly import products.

This article aims to describe the detection requirements of STL library for key components such as CPU, Clock, FLASH, RAM, etc., to ensure that the user's product system has sufficient reliability and security. Referring to Table H.11.12.7 of IEC60730, it includes the components that need to be tested, related faults, and testing plans. The table is organized as follows:

Table 1-1 IEC60730 Table H.11.12.7 Description

Components that need to be tested		**Fault/Error**	**Fault Classification**	**Nsing STL Library**	**Overview of Test Plan**
1.CPU	1.1 register	Stuck at	MCU related	YES	Write relevant registers and check
	1.3 program counter	Stuck at	MCU related	YES	If the PC runs away, start the watchdog reset
2. interrupt		No interruptions or frequent interruptions	Application related	NO	Calculate the number of interrupt occurrences
3. clock		Wrong frequency	MCU related	YES	Use HSI to measure LSI clock frequency
4. memory	4.1 Non-volatile memory	All single bit errors	MCU related	YES	FLASH CRC integrity check
	4.2 Volatile memory	DC fault	MCU related	YES	1. SRAM March C Test 2. Stack overflow detection
	4.3 Addressing (related to non-volatile and volatile memory)	Stuck at	MCU related	YES	FLASH/SRAM testing is included

NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

5. Internal data path	5.1 data	Stuck at	MCU related	NO	Only for MCUs that use external memory, single-chip MCUs are not required
	5.2 addressing	bad address	MCU related	NO	
6. External communication	6.1 data	Hanming distance 3	Application related	NO	Add verification in data transmission
	6.2 addressing	bad address	Application related	NO	
	6.3 time sequence	Wrong timing sequence	Application related	NO	Calculate the number of communication events
7.input/output	7.1 Digital I/O	Error defined in H27	Application related	NO	None
	7.2 Analog input/output	Error defined in H27	Application related	NO	None

NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

2. Terminology and Abbreviations

2.1 Terms and abbreviations used in this document

Table 2-1 Terminology and Abbreviations

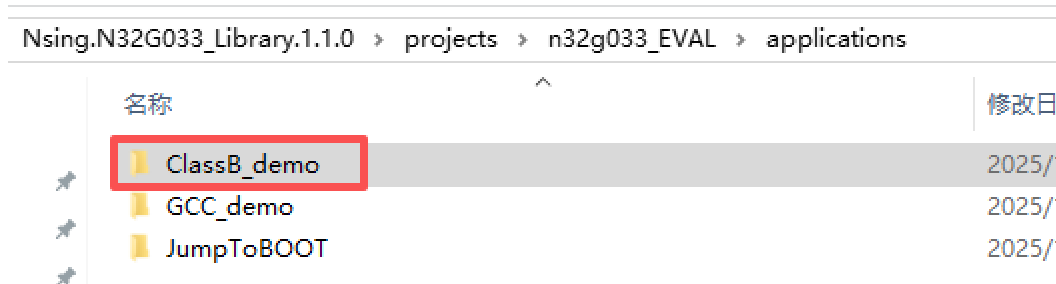
serial number	Terminology and Abbreviations	Instructions
1	N32	Nsing 32-bit chip
2	MCU	Microcontroller unit
3	CPU	Central Processing Unit
4	STL	Self checking library
5	RAM	Random Access Memory
6	CRC	Cyclic redundancy check
7	IO	Input/output interface
8	USARTx	Universal synchronous asynchronous receiver/transmitter, x indicates serial number, such as USART1
9	LSI	Low speed internal clock

NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

3. Software Package Architecture

The note code part of the application is released with the SDK, and the project directory is as follows (taking Nsing.N32G033_Library. 1.1.0 version as an example):

Figure 3-1 SDK application note location



Class B detection and diagnosis software package, based on the file structure of the N32 MCU chip firmware library, with the addition of integrated IEC60730 security detection related code. The overall structure framework of the software package folder is shown in the following table:

Table 3-1 Class B software package structure

Main Directory	**subdirectory**	**Remarks**
firmware	CMSIS	Kernel related drivers
	n32g033_std_periph_driver	N32G033 series peripheral driver
middlewares	N32_SelfTest_Library	N32 MCU STL Code Library
projects	n32g033_EVAL/applications/ClassB_demo	Integration example based on evaluation board, including Keil The IAR project

The specific N32 MCU product and evaluation board have been fully developed and tested in the following development environment and toolchain:

- Keil / MDK-ARM v5.34 version
- IAR- EWARM v8.50.1 version

3.1 Software Architecture Design Description

The STL software consists of chip peripheral driver libraries, Class B STL, and user applications. Class B STL is divided into two main parts: testing during the startup phase and periodic testing during the runtime phase.

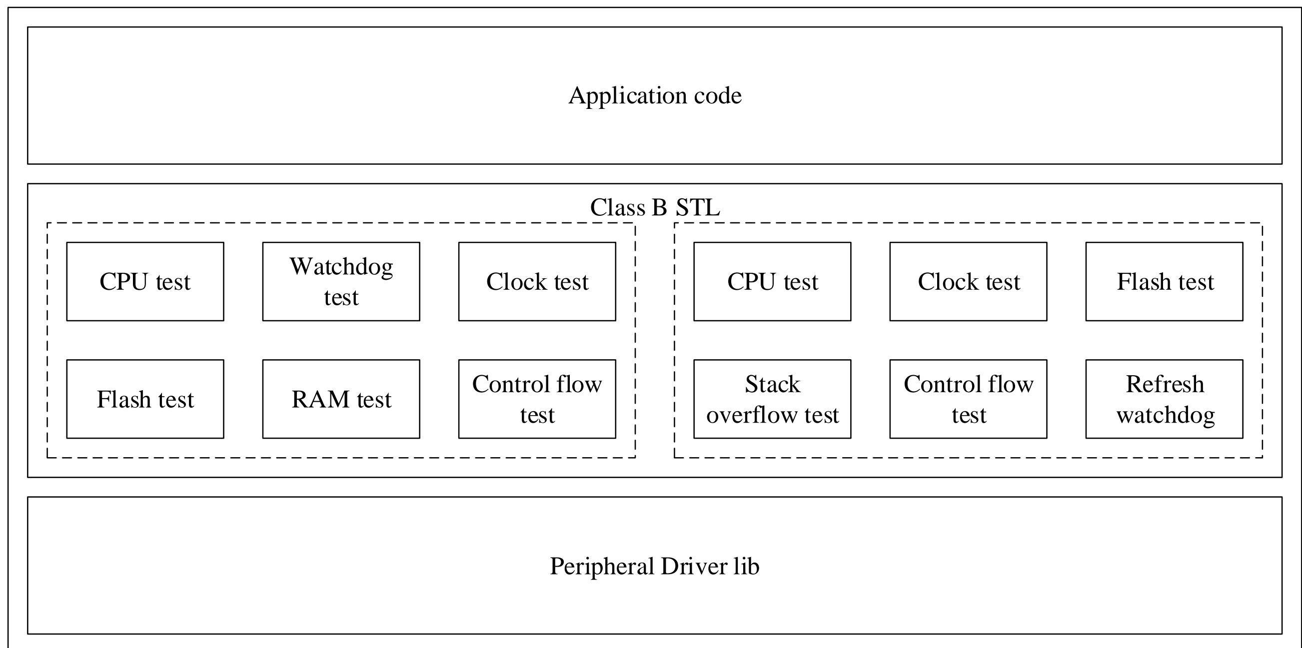
NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

3.2 Software Architecture Block Diagram

The overall block diagram of STL software architecture is shown in Figure 3-2.
Among them, the two parts in Class B STL include:

- CPU testing, watchdog testing, clock testing, Flash testing, RAM testing, and control flow testing during the startup phase.
- Periodic CPU testing, clock testing, Flash testing, stack overflow testing, control flow testing, and refresh watchdog testing during the runtime phase.

Figure 3-2 Software architecture diagram



NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

4. Software package components

The software package components include general parts and specific parts. The following tables list the corresponding files and their descriptions.

4.1 Overview of Software Components

The general STL software component file description is shown in the following table:

Table 4-1 General STL software file description

STL	Universal STL source and header files	
	files	description
Startup phase API	n32_cm0_STLstartup.c	Related code for STL process control during the startup phase
Runtime API	n32_cm0_STLmain.c	Related code for STL process control during runtime
STL source file	n32_cm0_STLclock.c	Related code for clock testing
	n32_cm0_STLcrc32.c	Related code for Flash CRC testing
	n32_cm0_STLtranspRam.c	Related code for RAM testing
	n32_cm0_STLcpurunKEIL.s	The relevant code for CPU and RAM testing during startup and runtime is written in assembly language, specifically for Keil And IAR
	n32_cm0_STLcpustartKEIL.s	
	n32_cm0_STLRamMcMxKEIL.s	
	n32_cm0_STLcpurunIAR.s	
	n32_cm0_STLcpustartIAR.s	
	n32_cm0_STLRamMcMxIAR.s	
header file	n32_cm0_STLparam.h	User defined header files for STL related functional variables
	n32_cm0_STLclassBvar.h	Header file for declaring Class B variables
	n32_cm0_STLclock.h	Header file for clock testing
	n32_cm0_STLcpu.h	Header files used for CPU testing
	n32_cm0_STLcrc32.h	Header file for Flash CRC testing
	n32_cm0_STLlib.h	The header file used for all source file includes, which is a collection of other header files
	n32_cm0_STLmain.h	Header file for STL process control during runtime
	n32_cm0_STLRam.h	Header files for RAM testing
	n32_cm0_STLstartup.h	The header file used for STL process control during the startup phase

4.2 Software Component Interface Definition

The interface definitions available for users to call in STL software package components are as follows:

Table 4-2 STL interface definition

No.	Name	Interface definition	Type	Remarks
1	STL_VerbosePORInit	Initialize USART for serial port printing and	void	The example uses UART1

NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

No.	Name	Interface definition	Type	Remarks
		IO pins used in testing		
2	STL_StartUp	Self checking process during the startup phase	void	This stage runs between reset and application (main function)
3	STL_InitRunTimeChecks	Initialize B-class variables and their reverse redundant corresponding variables, as well as initialize timers for clock frequency monitoring	void	SysTick generates 1ms time base
4	STL_DoRunTimeChecks	Self checking process during the operational phase	void	The call runs periodically. In the example, the call execution cycle is 20ms
5	SysTick_Handler	SysTick interrupt handling function	void	Includes RAM March check process, but with the same execution cycle as the self-test process during runtime (e.g. 20ms)
6	FailSafePOR	Error handling function	void	Provide three error handling methods: 1. Software reset 2. IWDG feeding dogs 3. WWDG feeding dogs

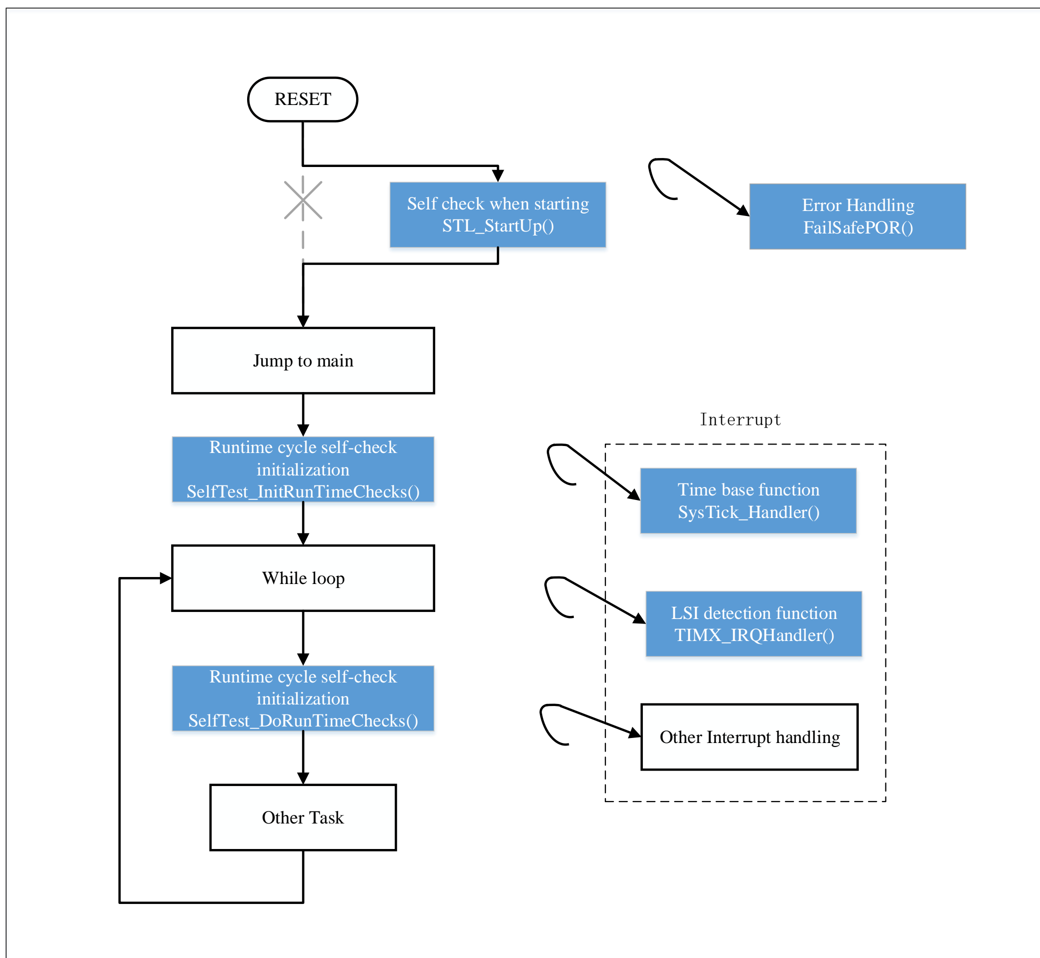
NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

5. Software Package Design

5.1 Module Detailed Design Description

The detection content of Class B software package programs is divided into two main parts: self check at startup and periodic self check at runtime. The overall flowchart is shown in the following figure. The blue diagram in the figure represents the additional parts that need to be added to execute Class B relative to the original application:

Figure 5-1 Software package flowchart



After MCU reset, the STL_StartUp() self-test program needs to be run before the application program is executed during the first check; After completing the self-test, enter the main function and call STL_InitRunTimeChecks() to initialize the configuration for periodic self-test before entering the main loop; In the systick interrupt, call STL_InterruptRunTimeChecks() to perform RAM detection and obtain the time base of periodic detection, and then call STL_DoRunTimeChecks() in the main loop to perform periodic self-test.

Note: If the self-test program takes too long during runtime, the self-test process can be split and periodic self-test can be performed using systick interrupts.

NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

In principle, to integrate the self checking module into the application, users need to provide the following steps:

- Perform a self check test before the user program starts
- Regularly perform runtime self checks set for specific time periods (related to safety time)
- When the application is running, set up independent and window watchdogs to prevent them from expiring (fine-tuning can be done by testing the actual running cycle through STL)
- Set the correct test area for RAM and Flash startup and runtime testing
- Pay attention to the erroneous results of testing and take appropriate actions to ensure the security of the system
- Prevent potential conflicts with application software (especially interrupts, DMA, CRC, etc.)
- Exclude all debugging functions unrelated to any security tasks and only use them for debugging or testing purposes

5.2 Module Detailed Design Description

Perform startup detection before the chip enters the main function from startup.

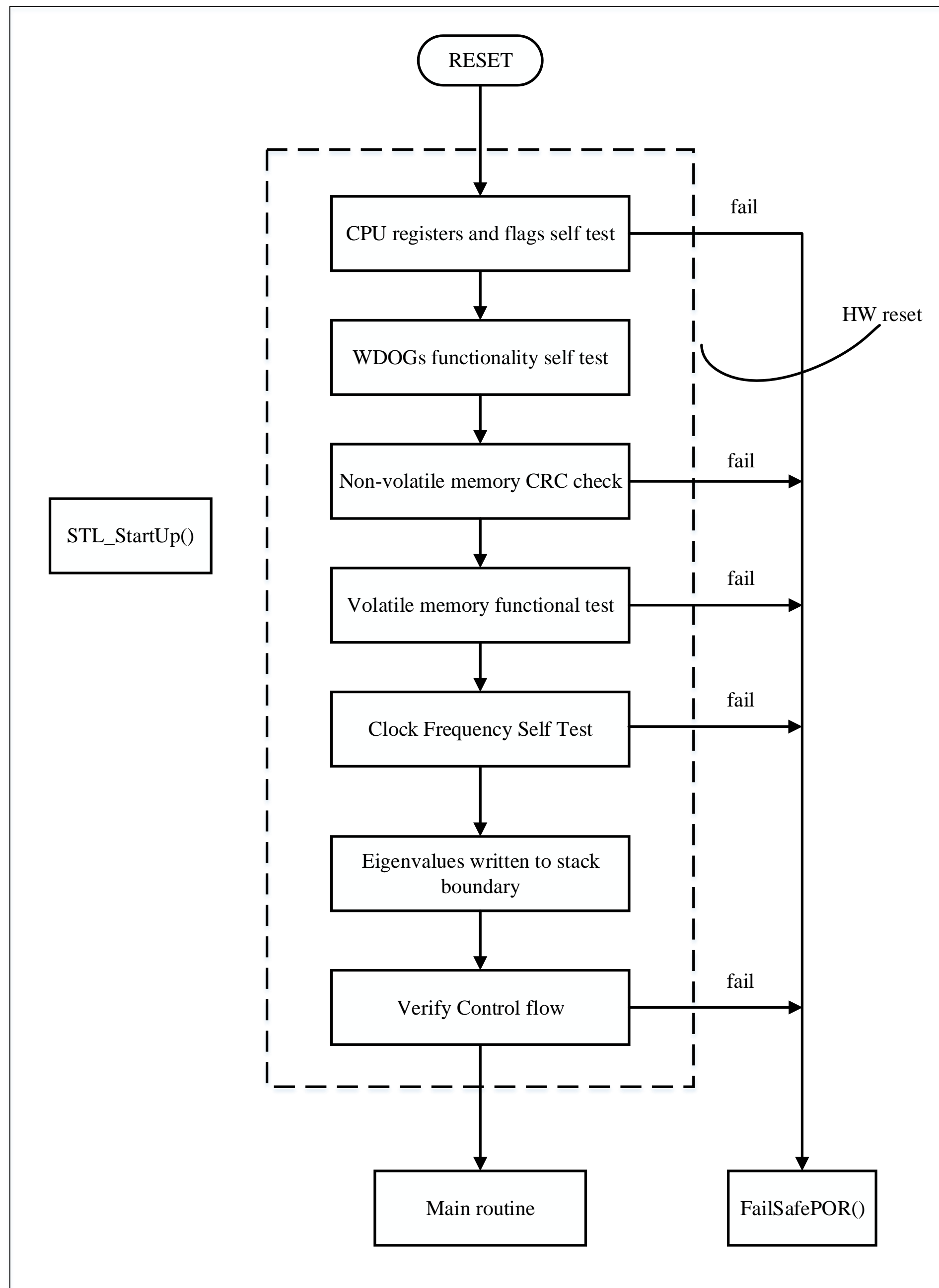
The self check during startup includes:

- CPU detection
- Watchdog detection
- Flash integrity detection
- RAM function testing
- Cross clock detection
- Write eigenvalues to the stack boundary
- Control flow detection

The following is a flowchart of the self check process during startup:

NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

Figure 5-2 Self check process diagram at startup



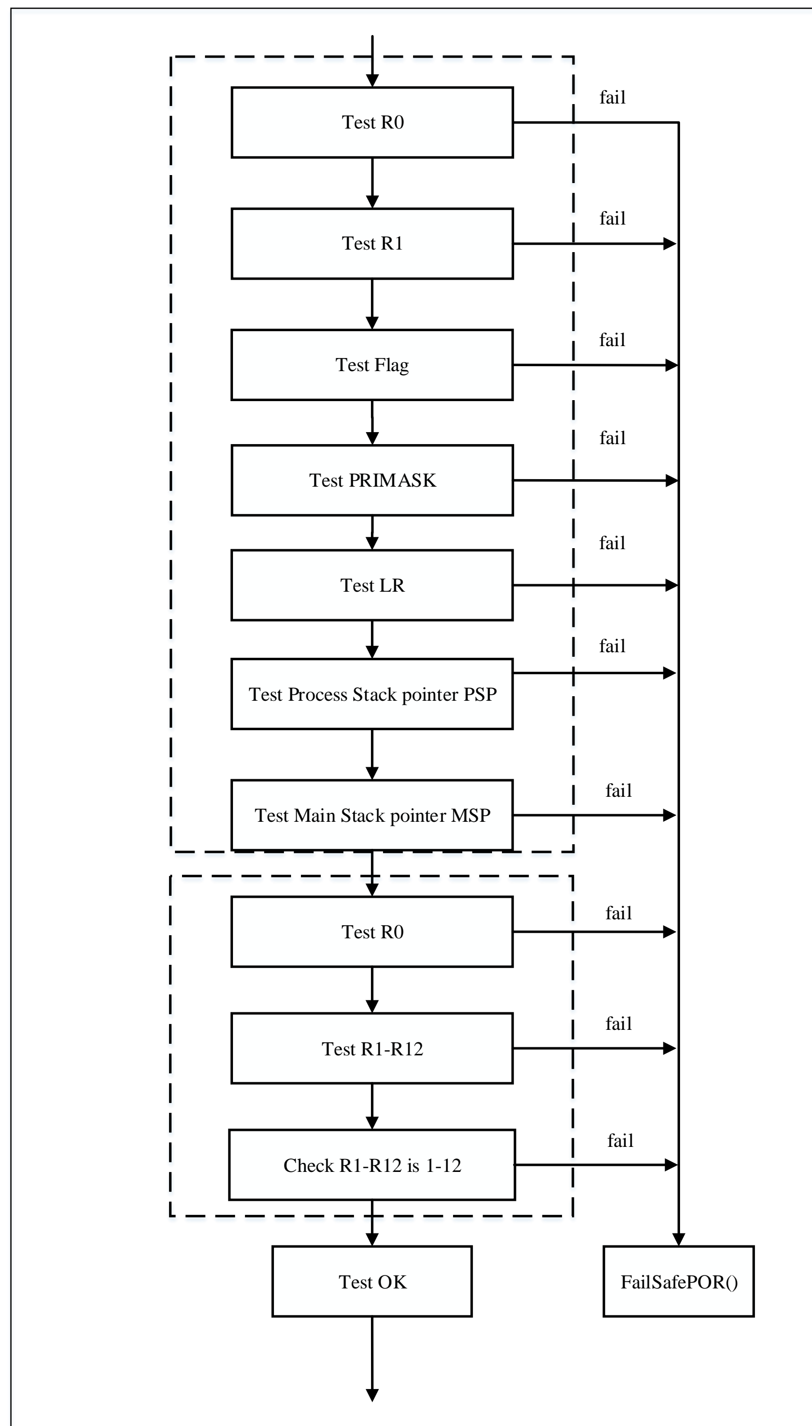
5.2.1 CPU startup detection

The CPU power on self-test mainly checks whether the kernel flags, registers, etc. are correct. If an error occurs, the fail safe handling function Failed Safe POR() will be called.

At startup, a self-test is performed on the Z (zero), N (negative), C (cargo), V (overflow), and Q (saturation) flags of R0~R12, LR, PSP, MSP, PRIMASK, FAULTRAMASK, BASEPRI registers, and APSR registers.

NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

Figure 5-3 CPU startup detection flowchart



5.2.2 Watchdog startup detection

Test and confirm that the independent watchdog and window watchdog can be reset correctly to ensure timely resetting during program execution to prevent jamming. The testing process is as follows:

1. If the IWDG reset flag is not set, proceed to step 2 to start IWDG testing. Otherwise, proceed to step 3;

NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

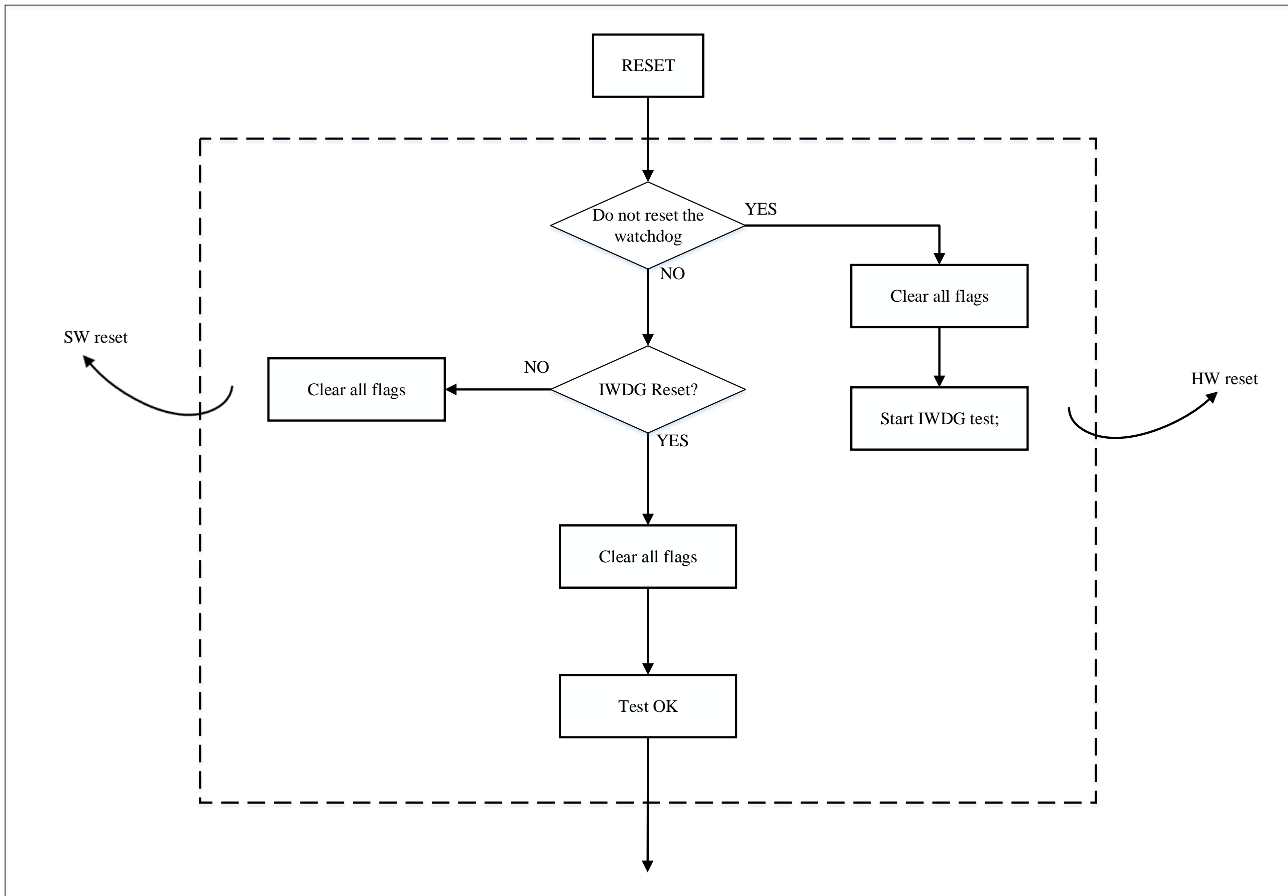
2. Clear all reset flags, start IWDG testing, enable IWDG, wait for reset without feeding the dog;

3. Check if the IWDG reset flag is set. If so, proceed to step 4. Otherwise, clear all reset flags and reset the software;

4. Clear all reset flags and pass the test.

The flowchart is as follows:

Figure 5-4 Block diagram of watchdog startup detection process



5.2.3 FLASH startup detection

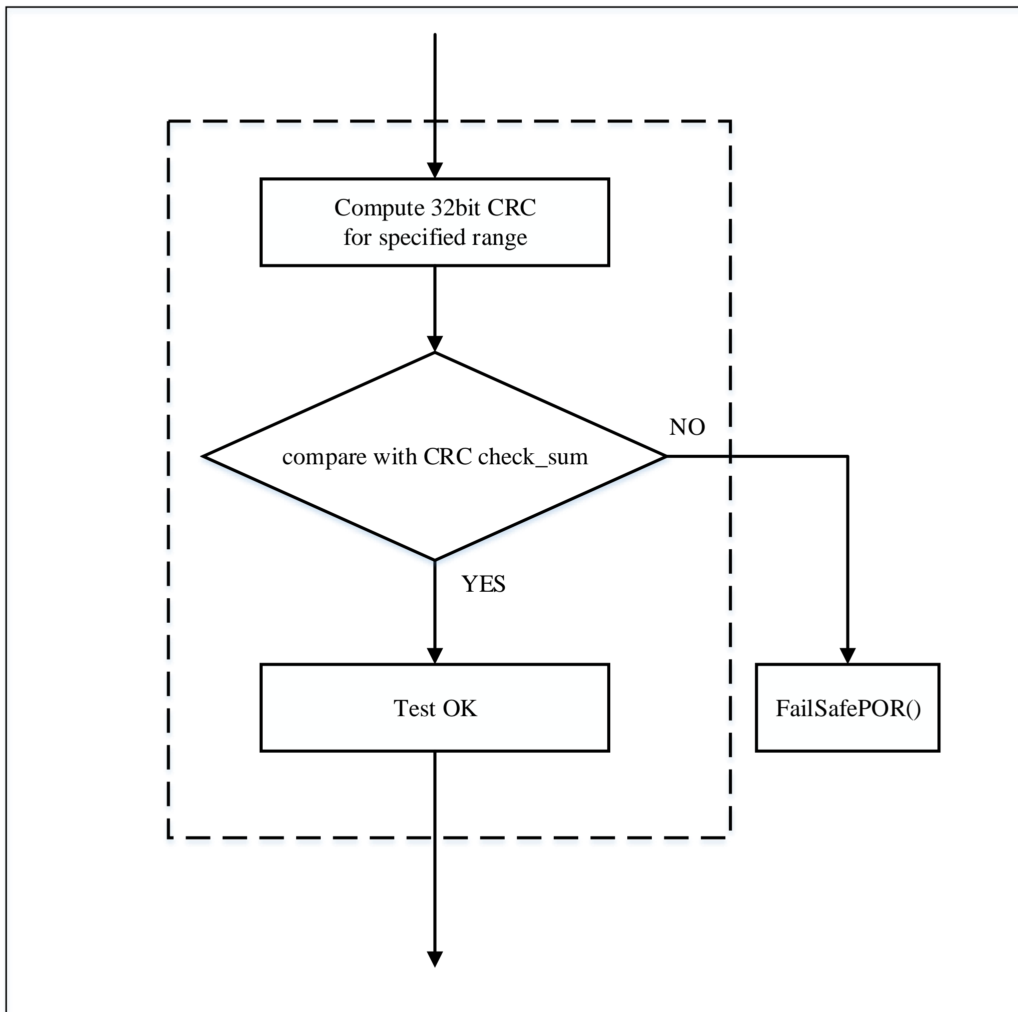
FLASH self-test is the process of calculating FLASH data using the CRC algorithm in a program, comparing the resulting value with the CRC value calculated during compilation and stored in a designated location in FLASH to confirm FLASH integrity.

At the same time, during the startup detection, the CRC32 data register will be checked in advance: after resetting the CRC, write 0x55555555 to the data register to determine if the check result is 0x85F89652, reset the CRC again, write 0xAAAAAAAA to the data register to determine if the check result is 0x42FC4B29.

The flowchart is as follows:

NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

Figure 5-5 Flowchart of FLASH startup detection process



The FLASH range for CRC calculation is configured based on the actual situation of the entire program, and it is necessary to ensure that the detection range is an integer multiple of 64 bytes.

The CRC value of Flash needs to be downloaded to the chip Flash along with normal application code during debugging or burning, so it needs to be added to the HEX compiled and generated in the integrated development environment (IDE). Below are how to configure Keil and IAR respectively.

IAR configuration:

The IAR configuration options support CRC calculation. Simply configure the parameters, and the compiled file will automatically add the CRC check result check_stum value to the selected FLASH calculation range.

Customers can configure IAR project options and SelfTest.iCF files based on actual applications, as shown in the following examples:

The CRC calculation range is 0x8000000-0x800FEFF, and the storage location for the inspection results is 0x800FF00

NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

Figure 5-6 IAR CRC Calculation Configuration Example

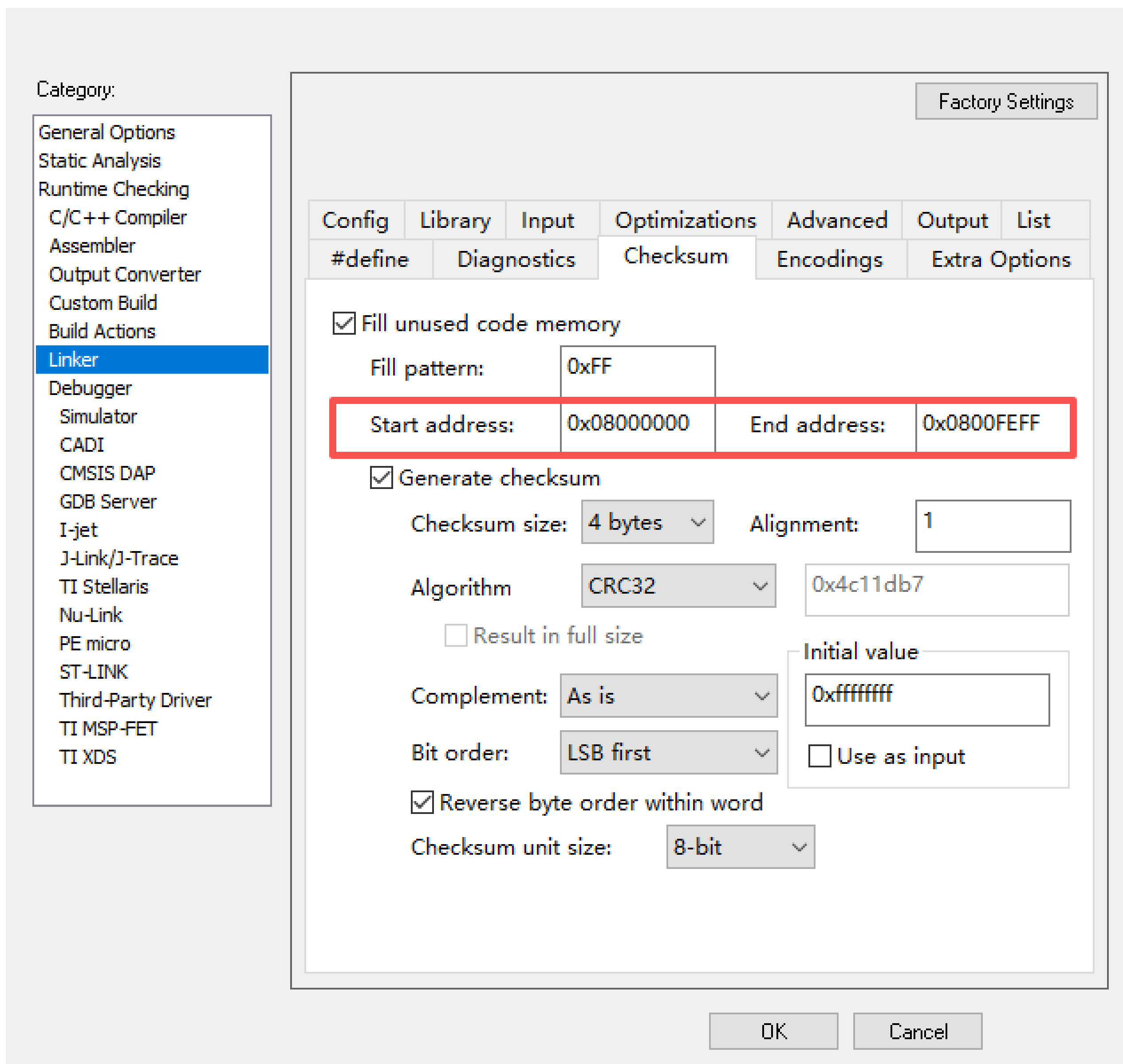


Figure 5-7 SelfTest.icf file

```
SelfTest.icf
1  /*###ICF### Section handled by ICF editor, don't touch! ****/
2  /*-Editor annotation file-*/
3  /* IcfEditorFile="$TOOLKIT_DIR$\config\ide\IcfEditor\cortex_v1_0.xml" */
4  /*-Specials-*/
5  define symbol __ICFEDIT_intvec_start__ = 0x08000000;
6  /*-Memory Regions-*/
7  define symbol __ICFEDIT_region_ROM_start__ = 0x08000000;
8  define symbol __ICFEDIT_region_ROM_end__   = 0x0800FF03;   /* Modify according to needs,Contains crc results */
9  define symbol __ICFEDIT_region_RAM_start__ = 0x20000000;
10 define symbol __ICFEDIT_region_RAM_end__   = 0x200017F0;   /* Modify according to needs */
```

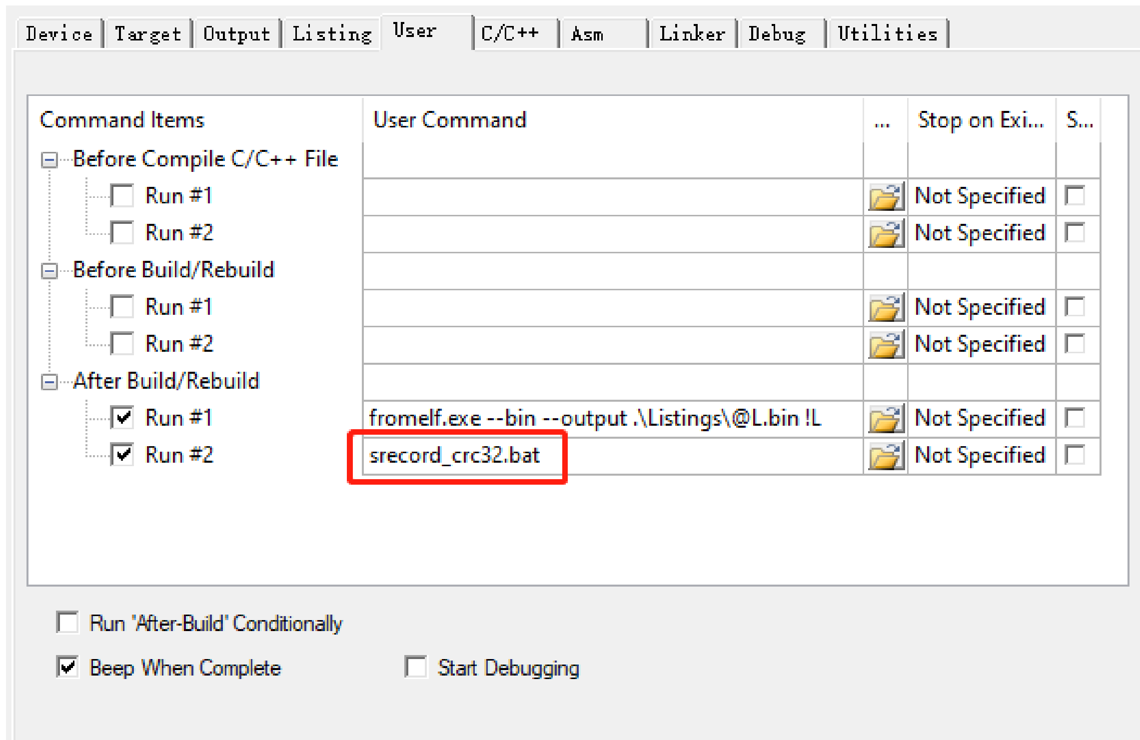
Keil configuration:

Keil's configuration is quite complex, and ARM recommends using third-party software SRecord for CRC testing of ROM Self Test in MDK-ARM.

NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

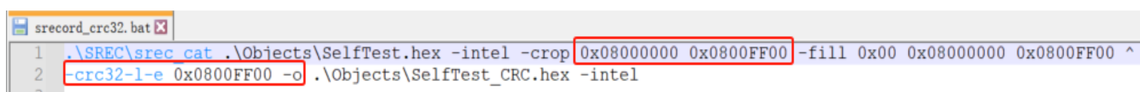
According to the engineering configuration, after compilation is completed, the script file 'srecord_crc32.bat' will be called. Using the software 'Srec_cat. exe', the data in the SelfTest.Hex file generated by Keil compilation will be subjected to CRC calculation to generate the CRC check result, which will be added to the specified location to obtain a new SelfTest_CRC.Hex file:

Figure 5-8 Keil CRC Calculation Configuration Example



Use Notepad or other tools to open the srecord_crc32.bat file, and the customer can modify the following content according to actual application. The CRC calculation range is 0x8000000-0x800FEFF, and the storage location of the inspection results is 0x800FF00:

Figure 5-9 srecord_crc32.bat file



The range for calculating CRC in the program is configured according to the n32ucm0_STLparam.h file and remains consistent with the above configuration:

NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

Figure 5-10 n32_cm0_STLparam.h file

```
n32_cm4_STLparam.h
121
122  #ifdef __CC_ARM    /* KEIL Compiler */
123
124    /* This is the KEIL compiler entry point, usually executed right after reset */
125    extern void __main( void );
126
127    /* Constants necessary for Flash CRC calculation (ROM_SIZE in byte) */
128      /* byte-aligned addresses */
129    #define ROM_START ((uint32_t *)0x08000000uL)
130    #define ROM_END   ((uint32_t *)0x0800FF00uL) /* Modify according to needs */
131    #define ROM_SIZE  ((uint32_t)ROM_END - (uint32_t)ROM_START)
```

And by configuring the crc_load.ini file as follows, KEIL can download and debug using the final generated SelfTest_CRC-hex file:

Figure 5-11 Keil settings

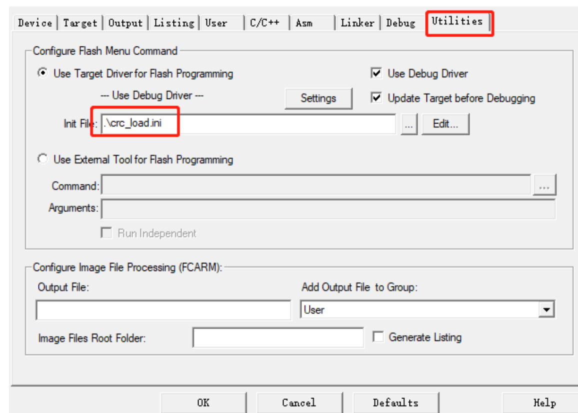


Figure 5-12 crc_load.ini file

```
crc_load.ini
1   LOAD "Objects\\SelfTest_CRC.hex"
```

5.2.4 RAM startup detection

SRAM detection not only detects errors in the data area, but also detects errors in its internal address and data path.

SRAM self-test adopts the March-C algorithm, which is an algorithm used for testing embedded chip SRAM and is part of security certification. Detect all ranges of SRAM during startup.

The test consists of 6 cycles, with values 0x55 and 0xAA alternately checking and filling the entire RAM word for word. The first 3 cycles are executed in address increment, and the last 3 cycles are executed in address decrement.

The detailed process is as follows. If the March X algorithm is used, steps 3 and 4 will be omitted:

1. Write 0x55555555 for the entire range, in ascending order of address;

2. Check if the entire range is 0x55555555, and then write 0xAAAAAAAA to the entire range in

NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

ascending order of address;

3. Check if the entire range is 0xAAAAAAAA, and then write 0x55555555 for the entire range, in ascending order of address;

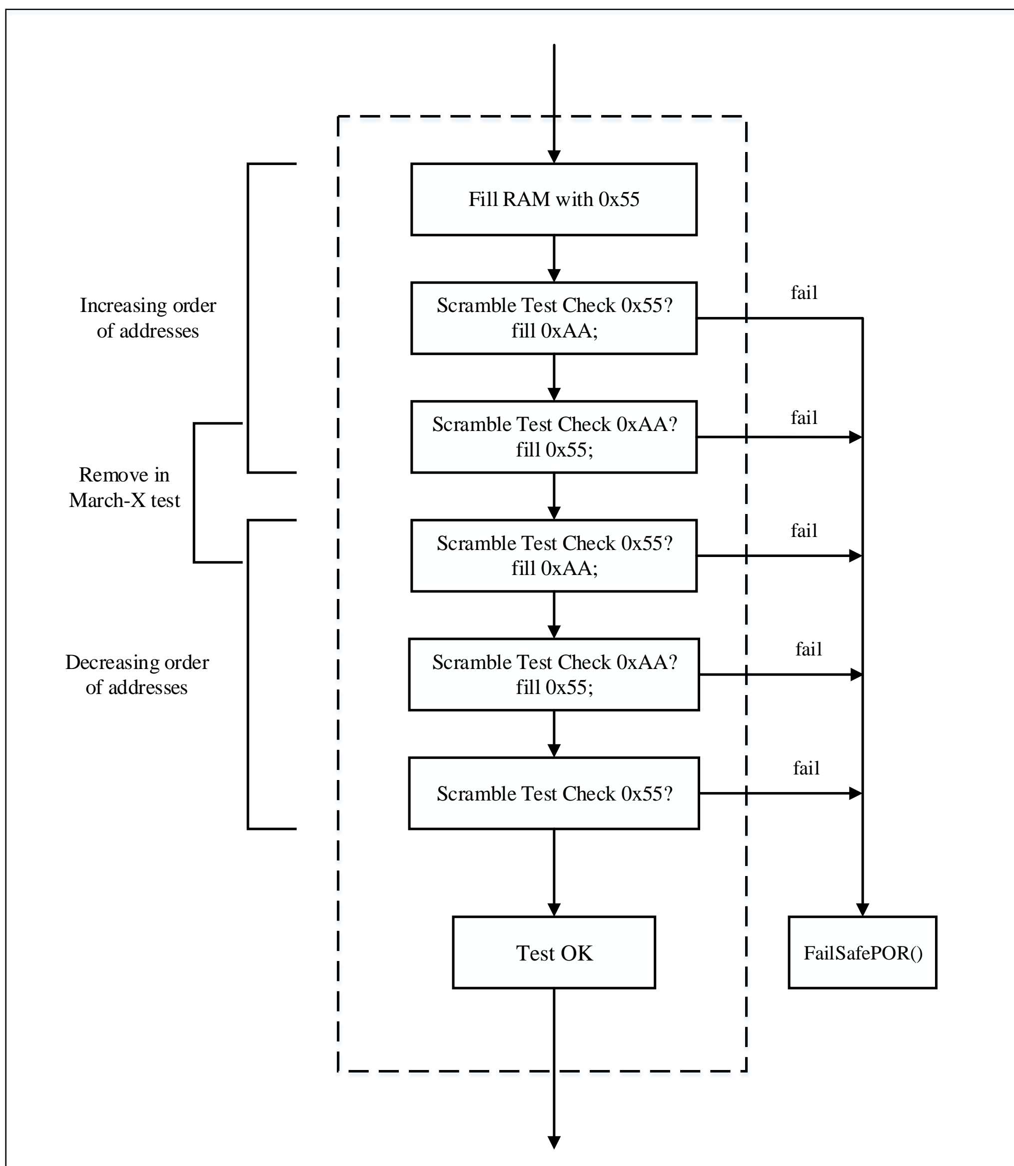
4. Check if the entire range is 0x55555555, and then write 0xAAAAAAAA to the entire range in descending order of address;

5. Check if the entire range is 0xAAAAAAAA, and then write 0x55555555 for the entire range, in descending order of address;

6. Check if the entire range is 0x55555555 and execute in descending order of address.

The flowchart is as follows:

Figure 5-13 RAM startup detection flowchart



It is also possible to perform scrambling operations on the test address order, as scrambling tests follow

NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

the physical address order, thereby preventing and identifying any crosstalk between adjacent physical storage units.

The basic physical unit is a model covering four words (one row), as shown in the table below: the numbering inside the cell represents the logical address, while other orders represent the physical layout, and the bold frame emphasizes the position where the logical order is scrambled.

Users can enable scrambling testing by configuring the ARTISAN parameter in the assembly bar.

Table 5-1 Example of Basic Physical Unit

	Physical address order --->			
Line--->	0	1	3	2
	4	5	7	6
	8	9	11	10
	12	13	...	

The RAM detection range at startup is configured through the following macro definition:

Figure 5-14 Range of RAM startup detection

```
151   /* Constants necessary for execution initial March test */
152   #define RAM_START ((uint32_t *)0x20000000uL)
153   #define RAM_END   ((uint32_t *)0x200017FFuL)  /* Modify according to needs */
```

5.2.5 Clock startup detection

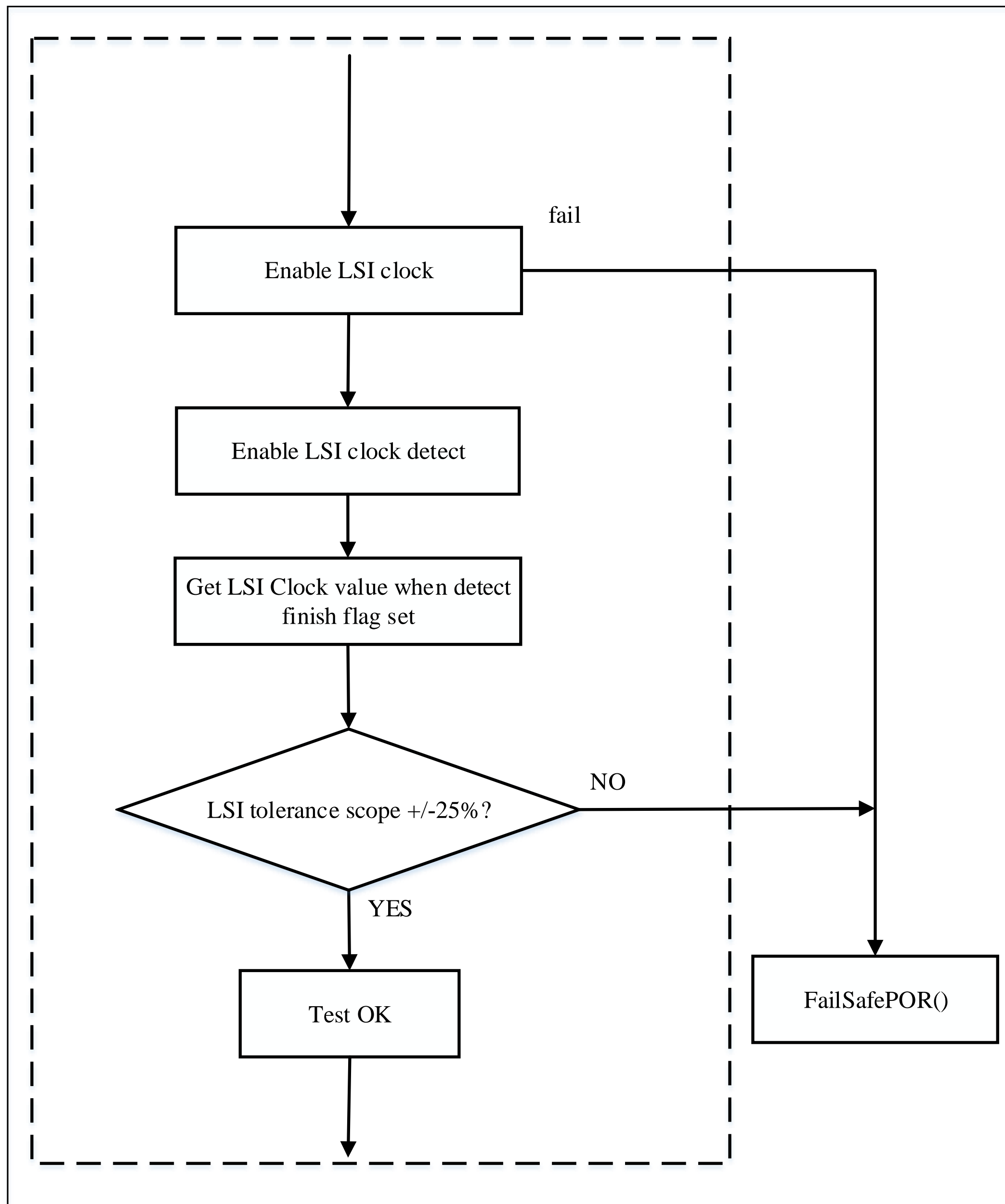
Built in HSI and LSI mutual detection function, the steps are as follows:

1. Enable LSI detection function;

2. When the LSIDETFF flag is set to 1, the detection ends. Read the register count value and compare it with the expected range value. If it exceeds the range, the test fails.

NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

Figure 5-15 Block diagram of clock startup detection process



5.2.6 Feature value writing stack boundary

Define an array adjacent to the bottom of the stack, and write the 0xAAAAAAAAuL, 0xBBBBBBuL, 0xCCCCCCCCuL, and 0xDDDDDDDDuL feature values to the array for easy detection of stack overflow during subsequent runs.

5.2.7 Detection during control flow startup

Control flow detection can check whether each ClassB detection function is called correctly (CALLER) and whether the function is executed correctly (CALLEE). The detection methods are summarized as follows:

1. Define two variables to indicate the progress of the control flow, set the initial values CtrlFlowCnt

NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

to 0x0000000 and CtrlFlowCntInv to 0xFFFFFFFF, and their initial states are reversed from each other;

2. Define two fixed values for each testing module to represent CALLER and CALLEE respectively, and assign different values to them;

3. Before calling a detection module, add a fixed value of CALLER to CtrlFlowCnt to indicate that the module has been called;

4. Enter the corresponding detection module and increase CtrlFlowCnt to the fixed value of CALLEE, indicating that the module is executing;

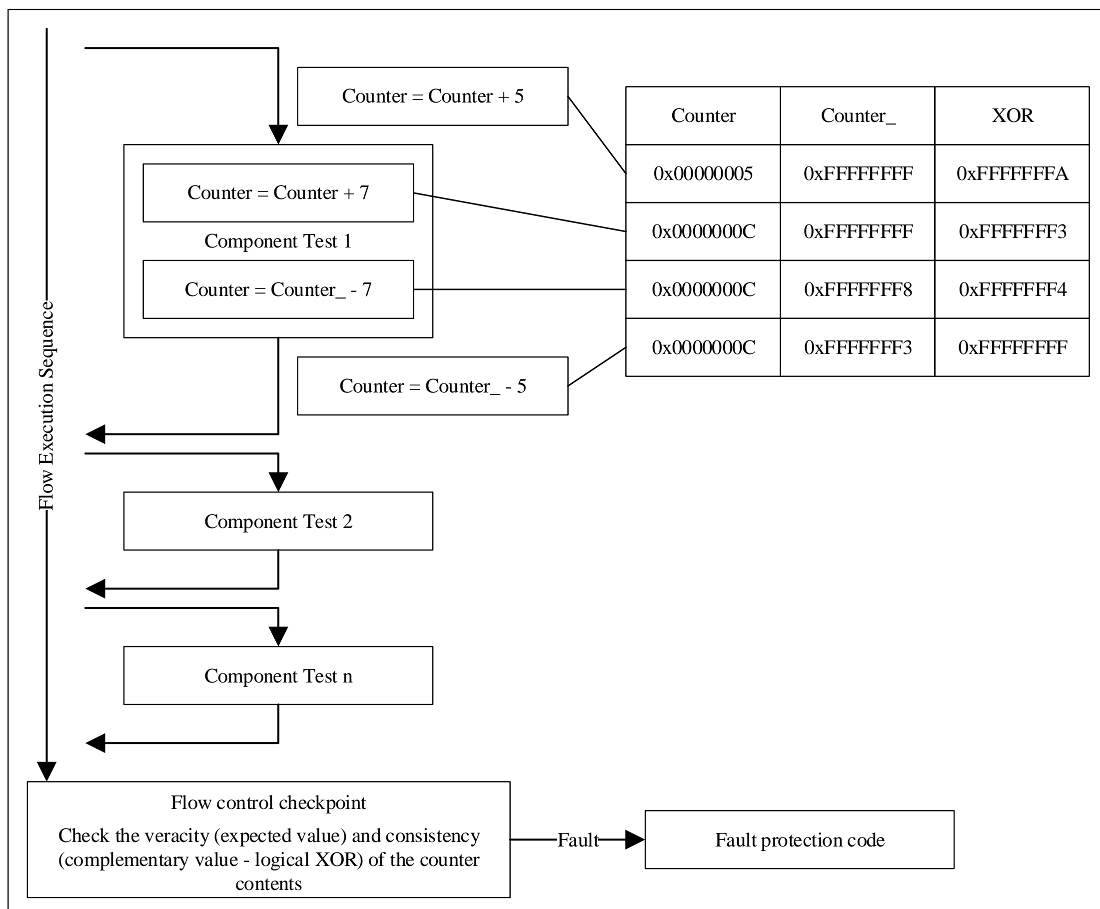
5. After executing the corresponding detection module, before exiting, decrease the fixed value of CALLEE by CtrlFlowCntInv to indicate that the module is executing correctly;

6. Complete the corresponding detection module. Before entering the next detection module, reduce CtrlFlowCntInv by the fixed value of CALLER to indicate that the module is called correctly;

7. Check if CtrlFlowCnt is the expected value and if it is still inverted with CtrlFlowCntInv. If so, it indicates that the corresponding detection module has been correctly called in the process and executed correctly.

The flowchart is as follows:

Figure 5-16 Block diagram of detection process during control flow startup

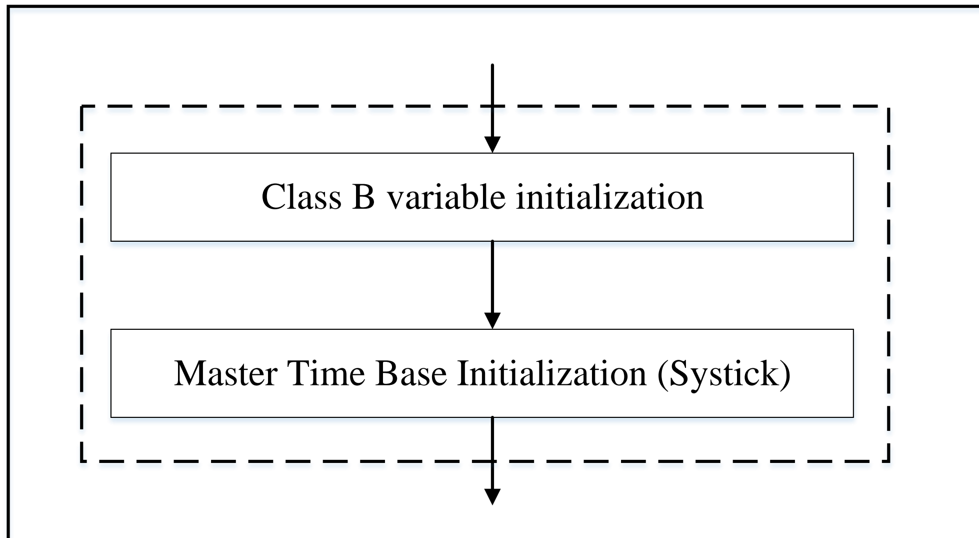


NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

5.3 Run time self-test initialization

If all self checks pass successfully during startup, the runtime self check must be initialized before the program enters the main loop (as shown in the figure below), and the periodic call of the runtime self check must be executed at the same time. The timing should be set correctly to ensure that the runtime test program is called periodically, thereby ensuring sufficient application running time.

Figure 5-17 Runtime self-test initialization diagram



5.4 Run time detection process

The runtime detection is based on SysTick as the time base and performs periodic detection, with the running cycle configured as needed.

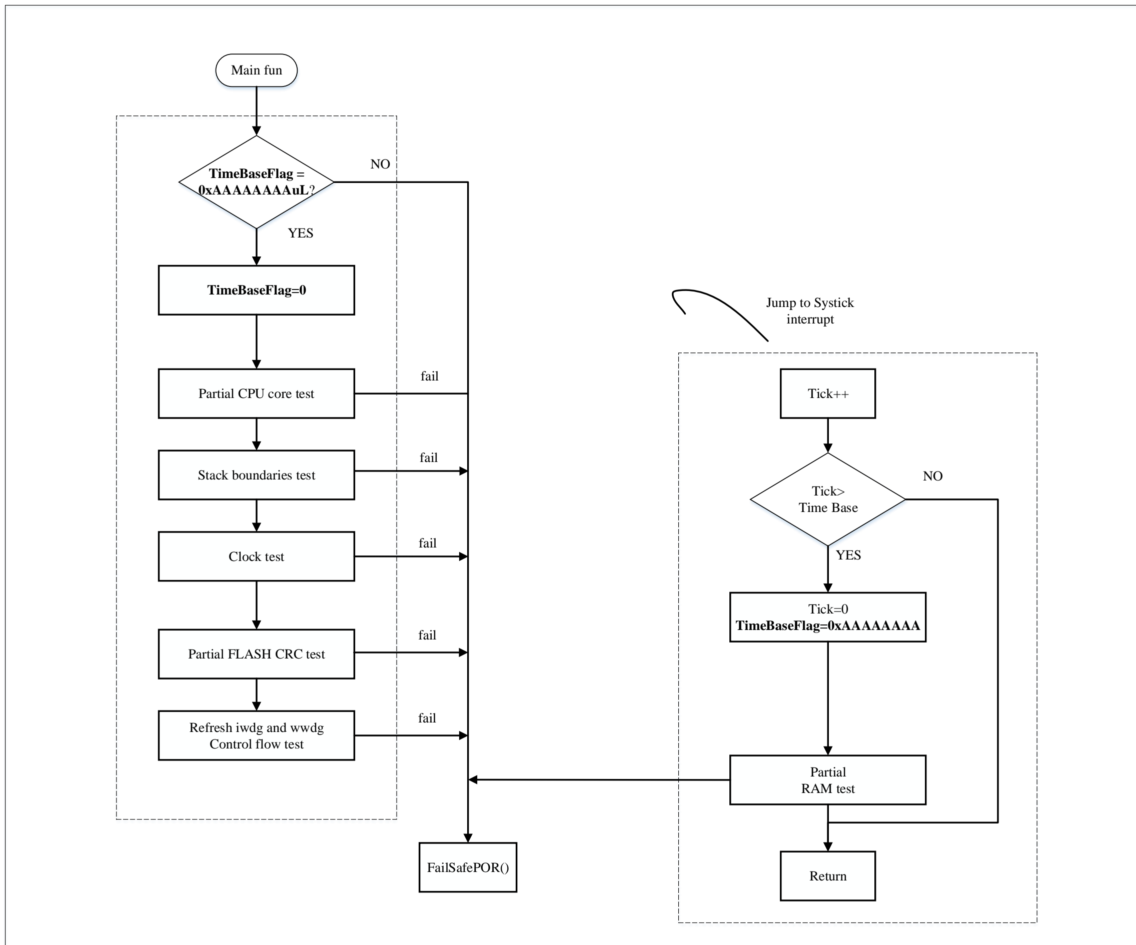
Periodic self-test during runtime:

- Local CPU core register detection
- Stack boundary overflow detection
- Clock operation detection
- Flash CRC segmentation detection
- Refresh during watchdog operation
- RAM March-C segmentation detection (performed during interrupts)

The runtime cycle detection process is as follows:

NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

Figure 5-18 Flow diagram of runtime cycle detection

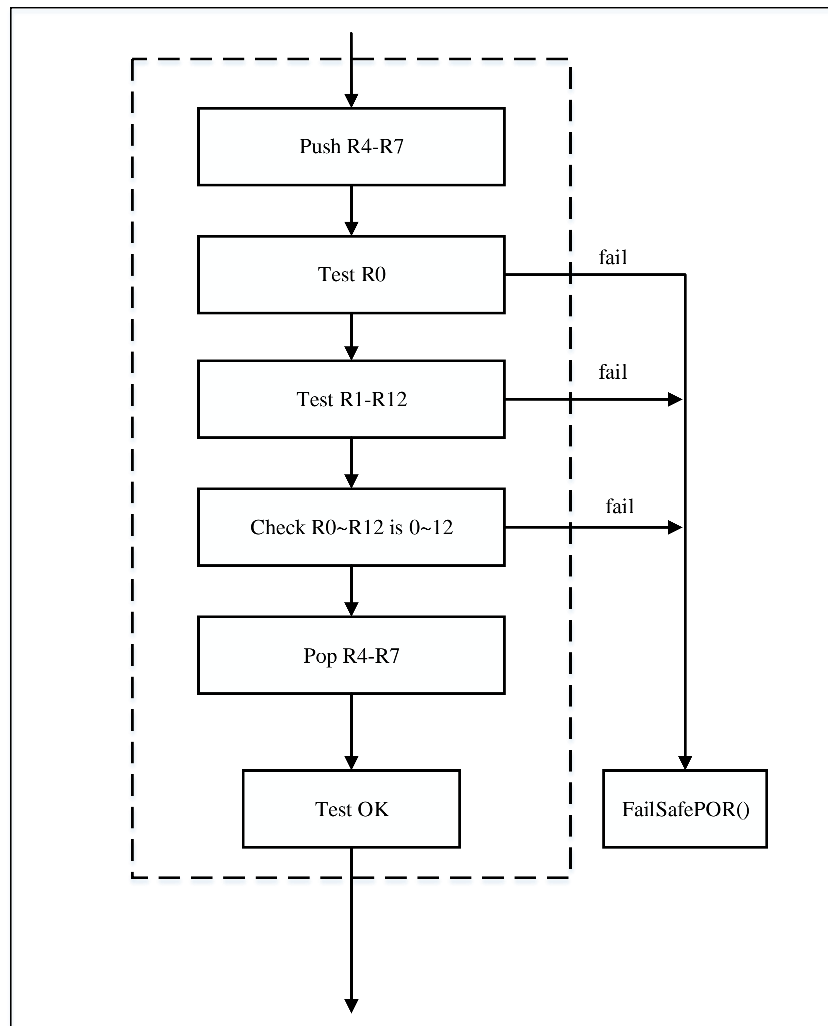


5.4.1 CPU runtime detection

The CPU runtime cycle self-test only detects the R0~R12 general-purpose registers, and the detection principle is similar to the self-test at startup, and will perform stack pushing and removal processing on the R4~R7 registers.

NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

Figure 5-19 CPU runtime detection flowchart



5.4.2 Stack Boundary Runtime Overflow Detection

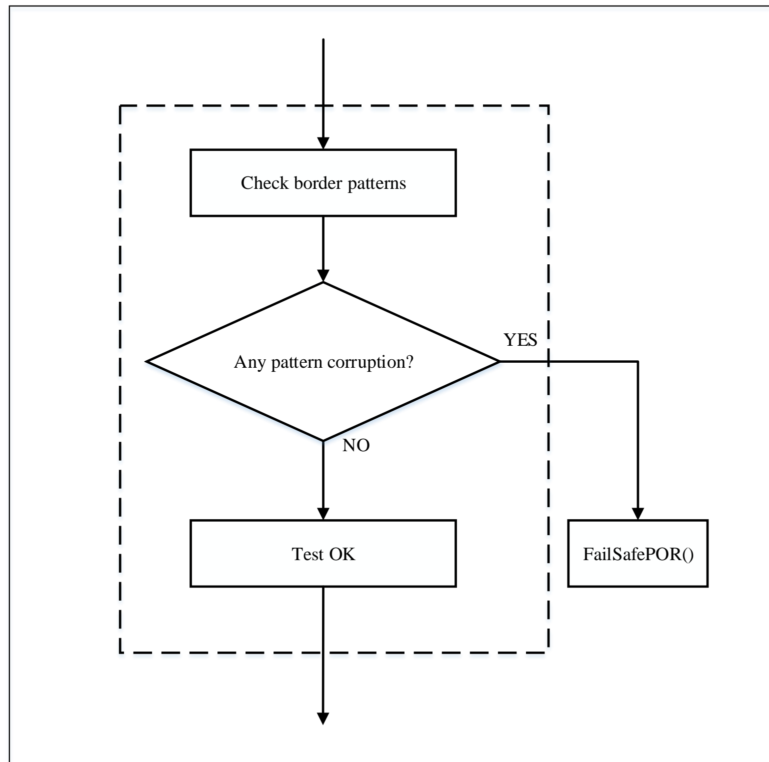
This test detects stack overflow by assessing the integrity of the pattern array data in the boundary detection area. If the original pattern data is corrupted, the test fails and the fail safe program is called.

The low address position immediately following the stack area is defined as the stack boundary detection area. Users must define sufficient areas for the stack and ensure that patterns are placed correctly.

The detection process is as follows:

NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

Figure 5-20 Stack boundary runtime overflow detection flowchart

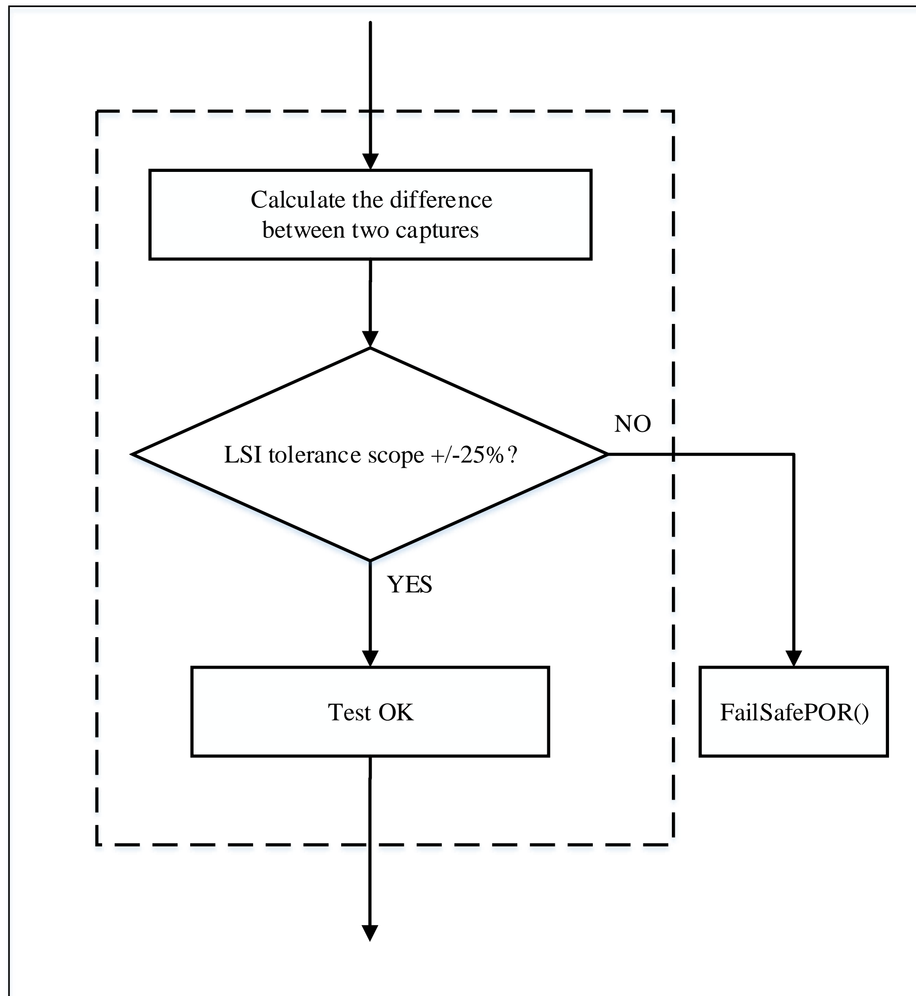


5.4.3 Clock runtime detection

The detection of runtime clock is similar to the detection of startup clock. The frequency of LSI is calculated by the difference between two captures. The process is as follows:

NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

Figure 5-21 Clock runtime detection flowchart

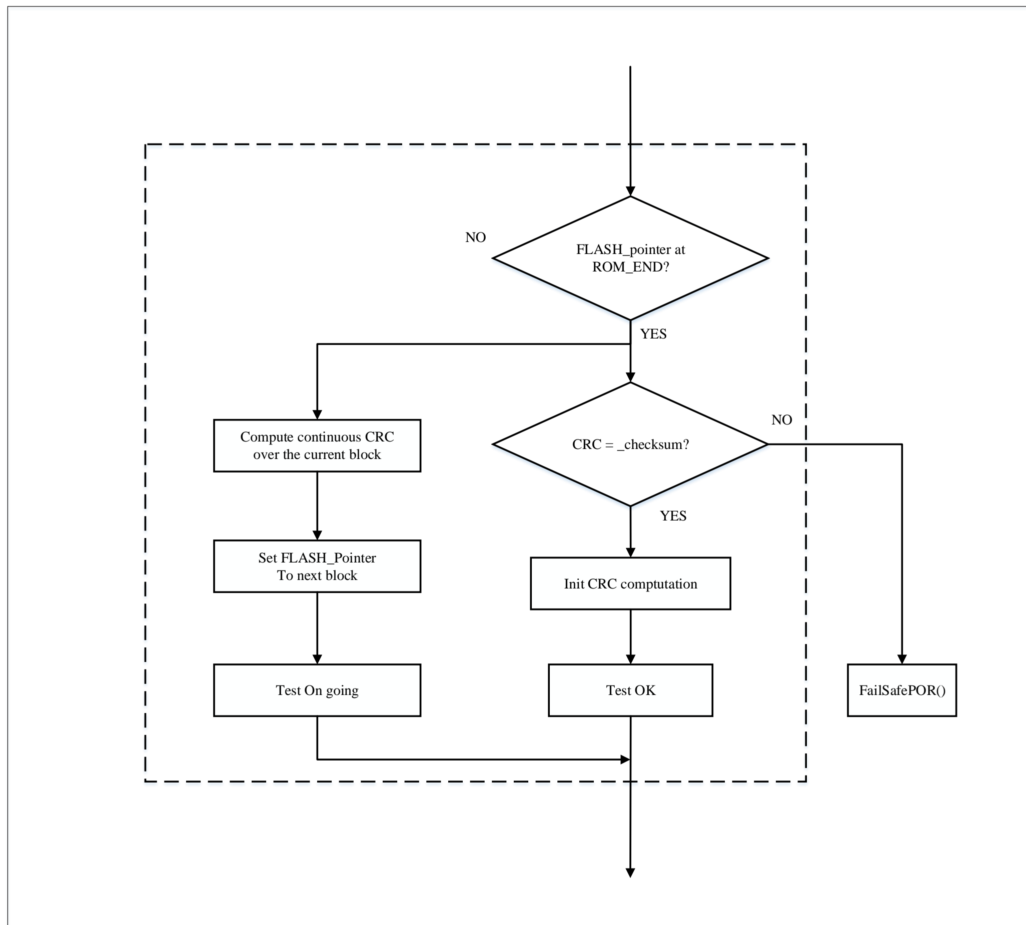


5.4.4 FLASH runtime detection

Perform self check of Flash CRC during runtime, as the detection range is different and the time consumption varies. If the CRC for all ranges is calculated at once, it may take too long and affect the normal execution of application code. Therefore, segmented CRC calculation can be performed based on the running cycle. When the last range is calculated, compare the CRC values. If they are not consistent, the test will fail.

NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

Figure 5-22 FLASH runtime detection flowchart



5.4.5 Watchdog runtime refresh

During runtime, it is necessary to regularly feed the dog to ensure the normal operation of the system. The part where the watchdog feeds the dog is placed at the end of STL_DoRunTimeChecks(), and users can choose whether to turn on the watchdog according to their actual application situation.

5.4.6 RAM runtime self-test

The RAM self-test during runtime is performed in the systick interrupt function, and the test range can be adjusted through the following macro definition. It should be noted that since the test includes adjacent words before and after the test area, appropriate margin should be reserved before and after the test range, and buffer blocks that temporarily store data and overflow chip RAM ranges should not be overwritten. It is also necessary to ensure that the detection range is an integer multiple of 16 bytes:

Figure 5-23 RAM runtime detection range

```
n32_cm0_STLparam.h
155    /* Constants necessary for execution of transparent run time March tests */
156    #define CLASS_B_START      ((uint32_t *)0x20000000uL)
157    #define CLASS_B_TESTSTART  ((uint32_t *)0x20000030uL)
158    #define CLASS_B_END        ((uint32_t *)0x200017F0uL) /* Modify according to needs: RAM_END - 0xF */
```

The distribution of RAM areas is shown in the following figure. KEIL divides RAM into reserved

NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

buffer area, Class B variable area, user variable area, stack boundary area, and stack area through SelfTest-N32XXXX.sct file, while IAR divides RAM into reserved buffer area, Class B variable area, user variable area, stack boundary area, and stack area through SelfTest.icf file.

Figure 5-24 RAM Area Distribution Map

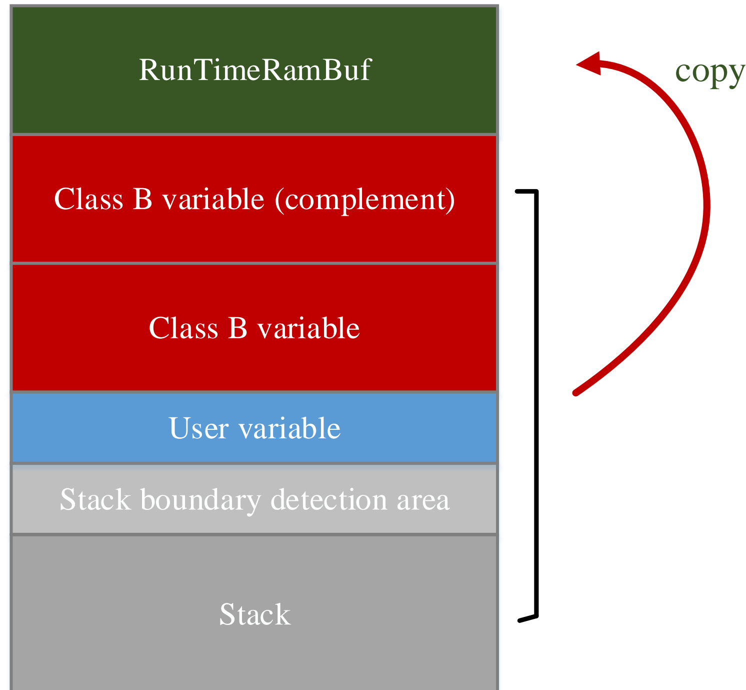


Figure 5-25 Example of Distributed File Loading

```
RAM_BUF 0x20000000
{            ; Run-time transparent RAM test buffer
  n32_cm0_STLstartup.o (RUN_TIME_RAM_BUF)
}

RAM_PNT 0x20000030
{            ; Run-time transparent RAM test pointer
  n32_cm0_STLstartup.o (RUN_TIME_RAM_PNT)
}

CLASSB 0x20000040 UNINIT
{          ; Class B variables
n32_cm0_STLstartup.o (CLASS_B_RAM)
}

CLASSB_INV 0x20000090 UNINIT
{              ; Class B inverted variables
n32_cm0_STLstartup.o (CLASS_B_RAM_REV)
}

RW_IRAM1 0x200000E0 UNINIT 0x00001300
{     ; RW data (Class A variables)
   .ANY (+RW +ZI)
}

STACK_NO_HEAP 0x200013E0 UNINIT 0x410  { ; Stack and magic pattern for stack overflow detection
  n32_cm0_STLstartup.o (STACK_BOTTOM)
  startup_n32g033.o (STACK, +Last) ;select right startup file
}
```

The testing process is as follows:

NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

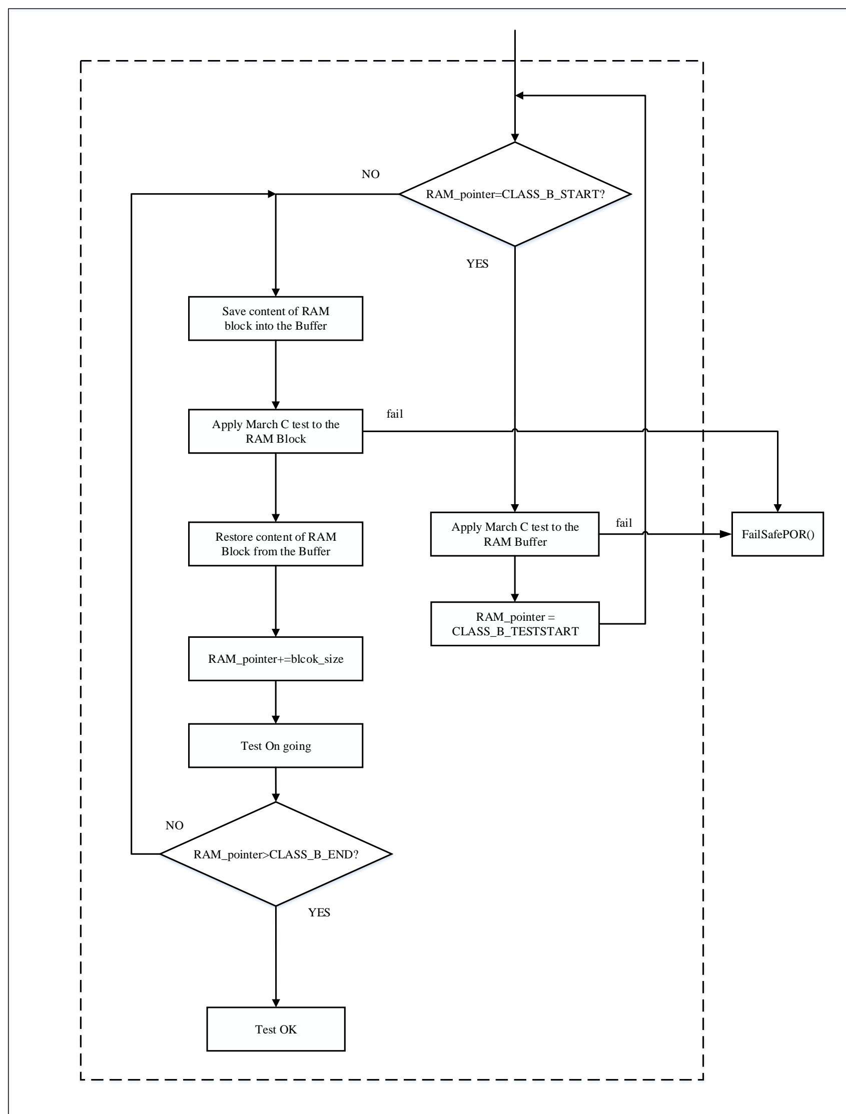
1. Perform March-C testing on the reserved buffer area to ensure that it is OK, and then perform testing on other areas in batches according to the interrupt cycle;

2. Store the data of the RAM block to be tested in a buffer block specifically designed for temporary data storage during the testing process;

3. Then, similar to detecting RAM at startup, the March-C algorithm is used to test the RAM block;

4. After the test is completed, restore the data stored in the buffer block to the test memory block (RAM block).

Figure 5-26 RAM runtime detection flowchart

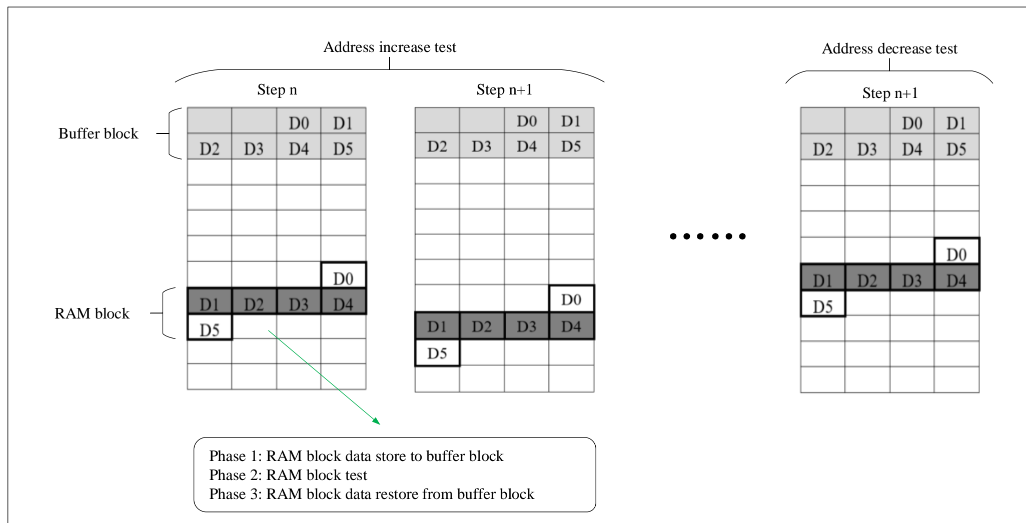


Detect an area of 4 words each time, but to ensure coupling fault coverage, the actual memory block tested each time also includes 1 adjacent word before and after the test area, for a total of 6 words.

The following figure describes the basic principle of fault coupling, with data numbers indicating the order of operations:

NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

Figure 5-27 Fault coupling



5.4.7 Control flow runtime detection

The runtime self-test ends by controlling the flow detection pointer program. The detection mechanism is similar to startup, where the RAM detection in SysTick interrupts and other detections in the main function are separated for control flow checking.

Initialize the variable CtrlFlowCnt to 0 and CtrlFlowCntInv to 0xFFFFFFFF. Add a fixed value to CtrlFlowCnt and subtract the same fixed value from CtrlFlowCntInv at each test step. At the end of the self-test, determine whether the sum of the two values still equals 0xFFFFFFFF.

NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

6. Precautions

During the user's use, if it is necessary to modify the software package related code according to the actual situation, it must be modified correctly according to the example project and combined with the relevant description in the previous text, otherwise it may cause detection failure. In addition, there are some IDE related issues that users need to pay attention to during development, such as:

1. Optimizing the compiler (setting the optimization level of the compiler) not only makes program analysis and debugging difficult, but may also lead to unexpected results. Therefore, it is strongly recommended not to optimize the Class-B part of the code during use.
2. When developing with different versions of IAR, unexpected results may occur in the program, so it is strongly recommended to develop based on the integrated example version environment.
3. During RAM detection at runtime, the global interrupt will be temporarily disabled and released after completion. Therefore, users need to be careful to prevent conflicts between application code and RAM detection. For example, for I2C modules, it is necessary to confirm data transmission and reception before performing RAM detection.

NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

7. Version history

Version	Date	Modify
V1.0.0	2025-11-26	Create a document

NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg

8. Notice

This document is the exclusive property of NSING TECHNOLOGIES PTE. LTD. (Hereinafter referred to as NSING). This document, and the product of NSING described herein (Hereinafter referred to as the Product) are owned by NSING under the laws and treaties of Republic of Singapore and other applicable jurisdictions worldwide. The intellectual properties of the product belong to NSING Technologies Inc. and NSING Technologies Inc. does not grant any third party any license under its patents, copyrights, trademarks, or other intellectual property rights. Names and brands of third party may be mentioned or referred thereto (if any) for identification purposes only. NSING reserves the right to make changes, corrections. enhancements, modifications, and improvements to this document at any time without notice. Please contact NSING and obtain the latest version of this document before placing orders. Although NSING has attempted to provide accurate and reliable information, NSING assumes no responsibility for the accuracy and reliability of this document. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. In no event shall NSING be liable for any direct, indirect, incidental, special, exemplary, or consequential damages arising in any way out of the use of this document or the Product. NSING Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, Insecure Usage'. Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, all types of safety devices, and other applications intended to supporter sustain life. All Insecure Usage shall be made at user's risk. User shall indemnify NSING and hold NSING harmless from and against all claims, costs, damages, and other liabilities, arising from or related to any customer's Insecure Usage Any express or implied warranty with regard to this document or the Product, including, but not limited to. The warranties of merchantability, fitness for a particular purpose and non-infringement are disclaimed to the fullest extent permitted by law. Unless otherwise explicitly permitted by NSING, anyone may not use, duplicate, modify, transcribe or otherwise distribute this document for any purposes, in whole or in part.

NSING Technologies Pte. Ltd.
Address: 20 Science Park, #03-15/16
Teleteck Park, East Wing, Singapore 117674
Email: sales@nsing.com.sg