# Introduction to Nuvoton IEC60730-1 Class B STL

Application Note for NuMicro® M0/M23/M4/8051/Arm9 Series

## Document Information

| | |
|---|---|
| **Abstract** | This application note introduces Nuvoton IEC60730-1 Class B Software Test Library (STL) consisting of low-level software routines. These routines implement basic requirements specified in IEC60730-1 Annex H MCU part of the standard. The user can add STL into existing application to meet the requirements from IEC60730-1. |
| **Apply to** | NuMicro® M0/M23/M4/8051/Arm9 Series. |





For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

# Table of Contents

# 1 Overview

IEC60730-1 is a safety standard for all home appliances sold in Europe. To accelerate certification process in products, Nuvoton provides users with the sample code consisting of low-level software routines. These routines implement basic requirements specified in Annex H of the IEC60730-1 standard. The hierarchy of developing the system is shown in the Figure 1-1. Users can add and modify this code into existing application to meet the requirements from IEC60730-1.

The main purpose of this application note and the associated software is to facilitate and accelerate user software development and certification processes for applications based on NuMicro series microcontrollers. The Software Test Library (STL) collects common sets of tests dedicated mainly to generic blocks of microcontrollers. The common part of STL package can also be reused for any other microcontrollers. The STL can be included into a final customer project, together with additional product specific tests and settings, for example, ADC and DAC tests.

With this application note, it introduces the methodology and techniques used in the STL. The provided examples show how to integrate the STL and the associated firmware in the application. The final implementation and functionality always has to be verified by the certification body at the application level.

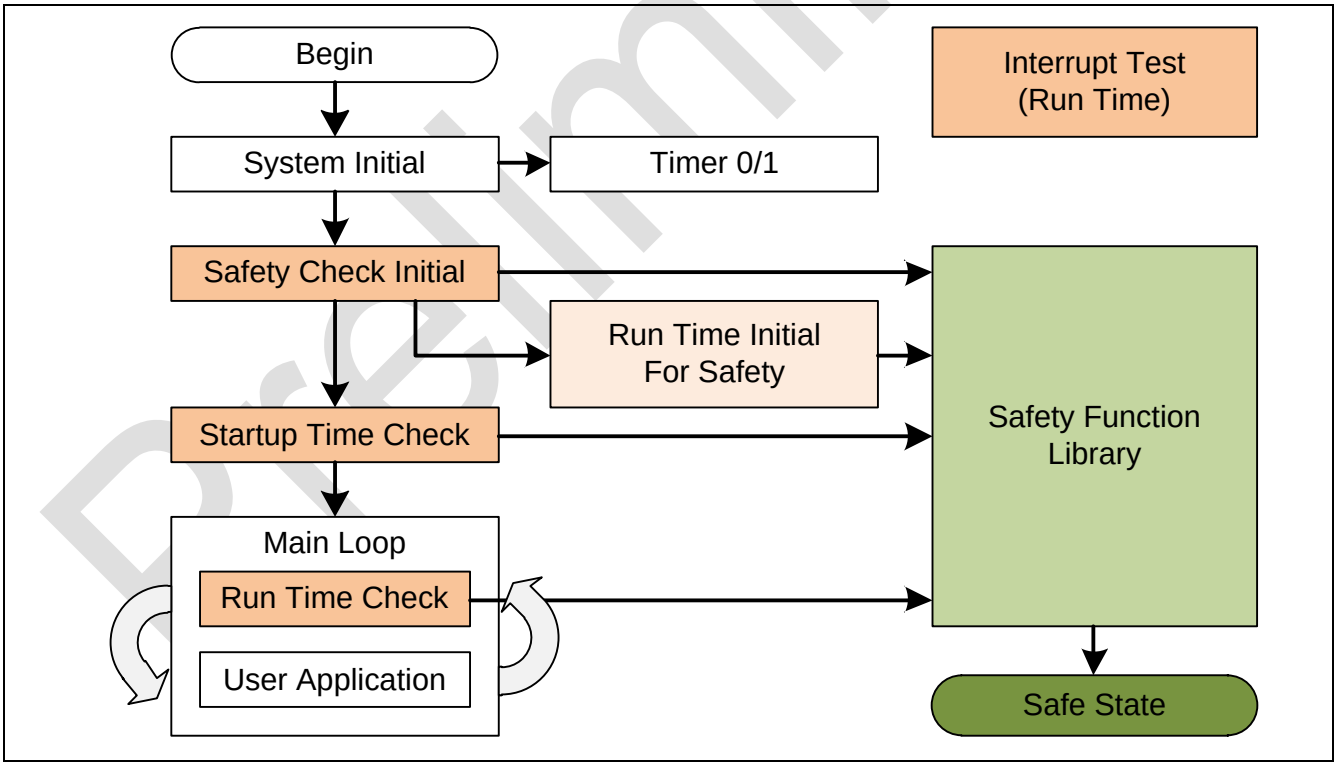



Figure 1-1 The hierarchy of developing the Nuvoton IEC60730-1 Class B STL

# 2 STL Structure Introduction

## 2.1 Safety Functions and Non-Safety Related Functions

For the safety section and non-safety section related functions could be categorized as below chart. The orange color groups are all related to safety functions and the other non-orange color groups are related to non-safety functions. The main loop in green color is the supervisory section, which controls the safety and non-safety sections.

| Safety Section Groups | Function Description |
|---|---|
| Safety Check Initial | The initial process for the Pre-Operation Self-Test (POST) at startup time and the Built-In Self-Test (BIST) at run time. |
| Startup Safety Check | Pre-Operation Self-Test (POST) test should be executed during system initialization. It is treated as a part of start-up procedure. |
| Run Time Safety Check | Built-In Self-Test (BIST) is executed periodically at run time. It cannot modify the content of current executing program, data or registers.<br>All the test functions must be done once in the POST test. |
| Interrupt Safety Check | For the risk time sensitive test functions The test function execution period should not longer than the predefined threshold value. |
| Safety Test Functions | A collection of the fundamental safety test functions, which can be called by the other safety function groups |
| Safe State | If the safety functions test fail, it will then report an error code and enter a specific safe state, which can do the cross ponding error handling and keep system safe. |

Table 2-1 Safety Functions List

| Non-Safety Section Groups | Function Description |
|---|---|
| System Initial | The initial process for the systems |
| User Applications | All functions related to applications only |
| **Supervisory Section** | **Function Description** |
| Main Loop | Controls the executions between safety and non-safety sections |

Table 2-2 Non-Safety Functions List

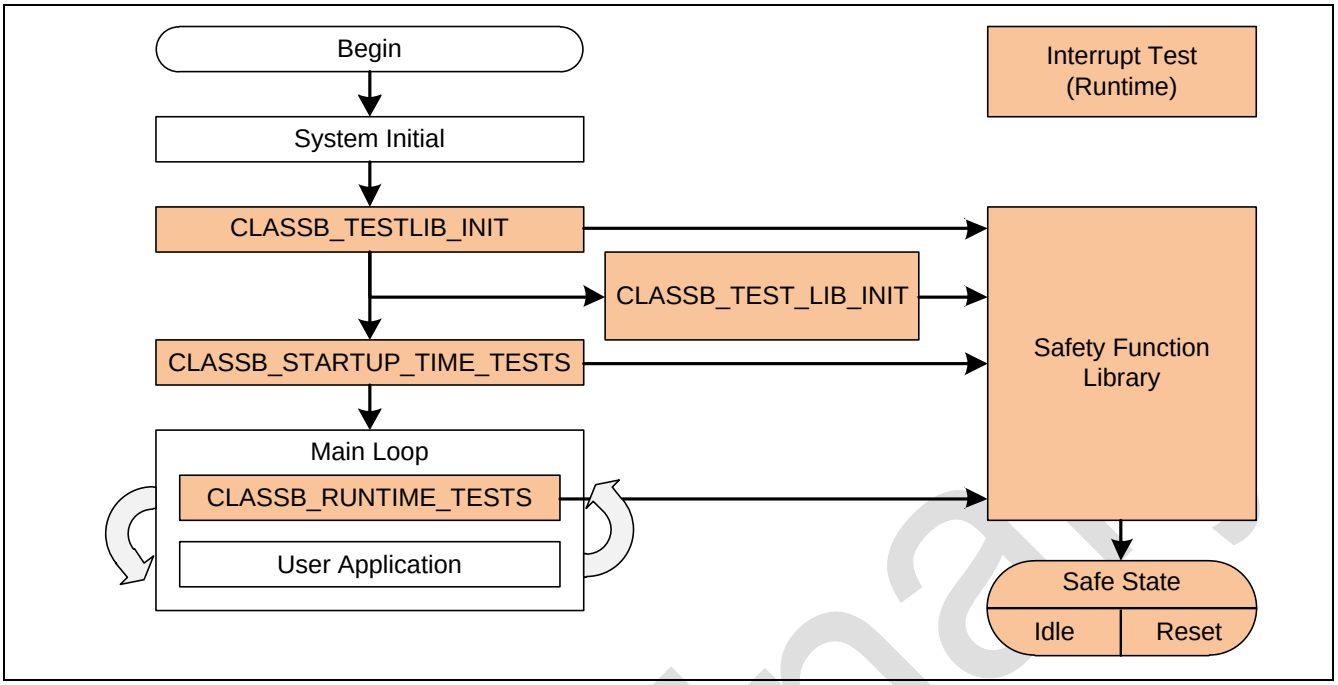



Figure 2-1 Safety and Non-Safety Related Functions

## 2.2 Test Items in Safety Functions

The STL consists of several kinds of test items, including CPU, Interrupt, Clock, Memory and I/O tests. The implemented software structures follow the Table H.1 – Software Class B from IEC60730-1 Annex H .All components to be tested by safety related functions are listed in Figure 2-2. These tests are executed during startup or run time stages or both. The test functions marked in solid line are released with this application note. Please unzip the STL and put the “STL Source Code” folder under the NuMicro series BSP root folder as shown in **Error! Reference source not found.**.

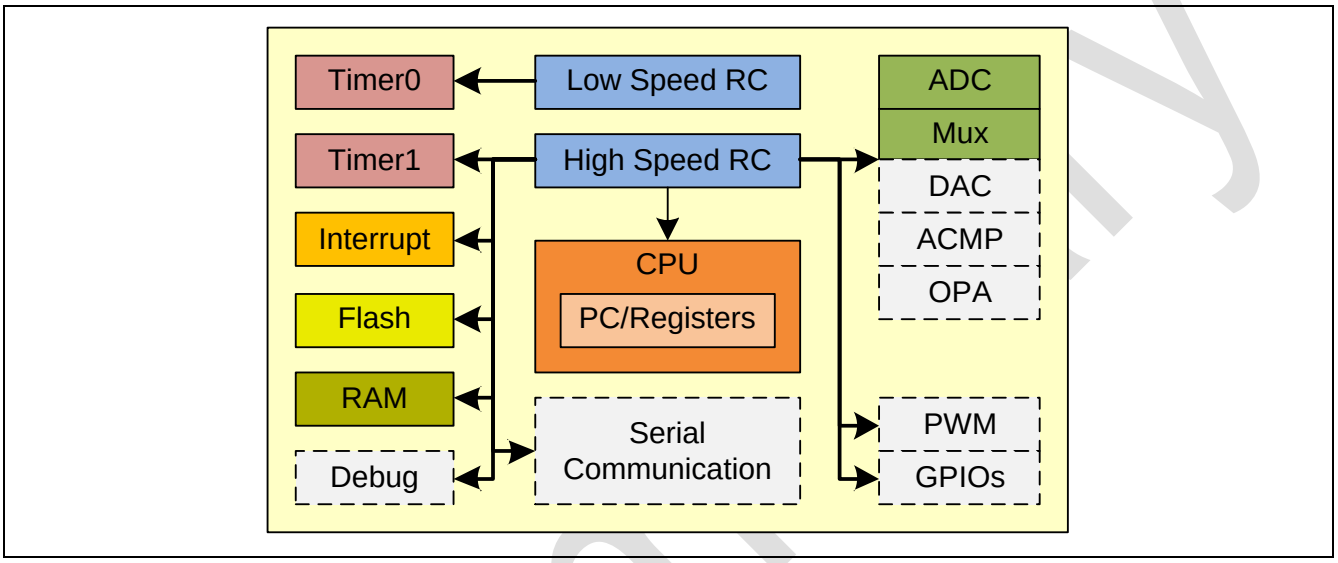



Figure 2-2 The Components Tested by Safety Related Functions

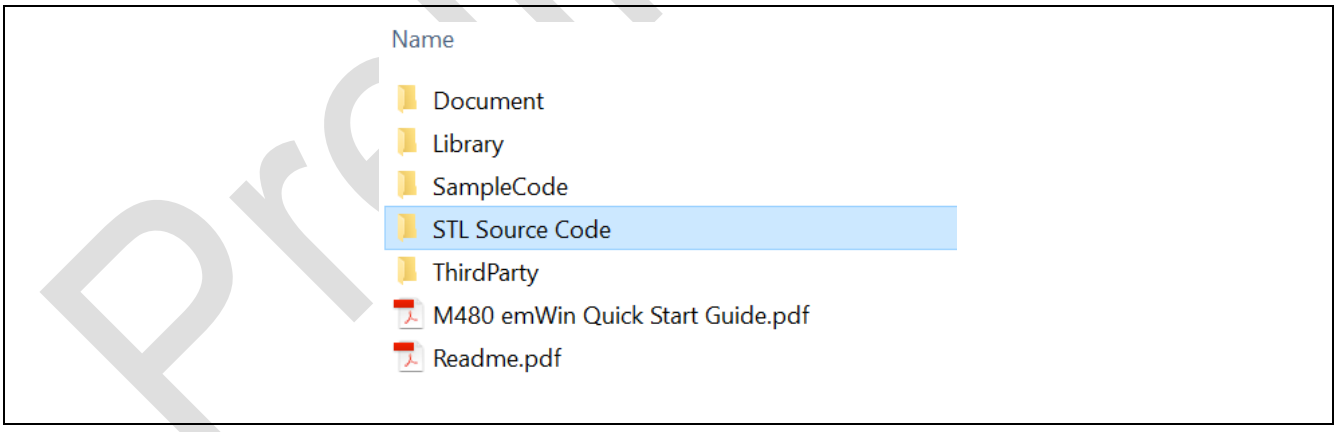



Figure 2-3 STL Package in BSP Folder

### 2.2.1 CPU: Registers, Program Counter, Stack

CPU with program counter (PC) and control/status registers is the core of the software. There are a number of registers in the kernel of main controller company with program counter and stack controls, which are essential items for the software operations of a main controller. These

are judged as critical items. This check is performed in both startup time and run time.

| Function Name | Test Target | Platform |
|---|---|---|
| CLASSB_Register_Test() | R0 ~ R12 | M0/M23/M4/ARM9 |
| | ACC, DPTR, R0 ~ R7 | 8051 |
| CLASSB_ProgramCounter_Test() | Program Counter (R15) | M0/M23/M4/8051/ARM9 |
| CLASSB_Stack_Test() | Stack register (R13) | M0/M23/M4/8051/ARM9 |

Table 2-3 CPU Test Target on Different Platforms

### 2.2.2 Interrupt: Timer0 and Timer1

Interrupts service the communication between software and hardware which are highly related to both safety test functions and normal user applications. The Fault Mode of the Interrupt includes no interrupt or too frequency interrupts, judged as a critical item. Besides the Clock generator, the controller uses a “Timer0” with a slow clock source and a “Timer1” with high speed clock source. By checking the counter ratio between these two timers, an Interrupt Fault could be verified if it is greater than a pre-defined threshold value which can be defined by a maximum error between these 2 clocks. This check is performed in run time.

| Function Name | Test Target | Platform |
|---|---|---|
| CLASSB_Interrupt_Clock_Test() | Timer0, Timer1 | M0/M23/M4/8051/ARM9 |

Table 2-4 Interrupt Test Target on Different Platforms

### 2.2.3 Clock: Timer0 and Timer1

Clocks play an important role in a controller system, which are fundamental components for others like CPU, Timer, Storage, Interrupt and as many peripherals (ADC/PWM, etc). There are mainly two clocks required by the safety related functions, one is a relative low speed clock and the other is a relative high speed clock. These two clocks are not only strictly limited using for safety functions but also can service the application requirements. Timer0 uses low speed clock and Timer1 uses high speed clock. The clock source of Timer0 and Timer1 can from an internal RC, external crystal or external clock input. This check is the same as interrupt check and also performed in run time.

### 2.2.4 Memory: SRAM and Flash

Memories, includes RAM and Flash, are used to store software program and data. RAM includes internal SRAM or other volatile memory. Flash includes internal APROM, LDROM, Data flash or other non-volatile memory. If any single bit fault occurs in these memories, the instructions decode and execution in a main controller could be wrong and result in a hard fault.

Therefore, the single bit fault of memory is judged as a Critical item.

APROM or Data Flash check can be performed by the checksum verification process which will compare the checksum calculation result from target memory with a pre-burned CHECKSUM value at the end of non-volatile memory (The checksum calculation should not include the area of pre-burned CHECKSUM value. The target memory range for testing are divided into small segments (32bytes for example) for run time testing in order not to impact the execution time of applications.

SRAM check can be performed by parity check, March X check or March C check, respectively dependents on the H/W character which is defined in the header file. All memories are divided into small segments (32bytes for example) for run time testing in order not to impact the execution time of applications.

| Function Name | Test Target | Platform |
|---|---|---|
| CLASSB_Flash_Test() | Flash memory (APROM) | M0/M23/M4/8051 |
| | DRAM | ARM9 |

Table 2-5 Flash Test Target on Different Platforms

| Function Name | Test Target | Platform |
|---|---|---|
| IEC60730_RamMarchC_Test() | Specified range SRAM | M0/M23/M4 |
| | Specified range internal RAM | 8051 |
| | Specified range DRAM | ARM9 |
| IEC60730_RamMarchX_Test() | Specified range SRAM | M0/M23/M4 |
| | Specified range internal RAM | 8051 |
| | Specified range DRAM | ARM9 |

Table 2-6 Memory Test Target on Different Platforms

### 2.2.5 I/O

GPIOs are usually required for external control or input status trigger, which can also service the functional safety required peripherals like ADC, PWM, ACMP, OPA. ADC pins for current measurement, PWM pins for driving motor, ACMP for over current protection, OPA/PGA for input current amplify and DAC provides the threshold value for ACMP over current protection. The functional safety required peripherals are application depended and outside the scope of this document. The STL only implemented ADC self-tests which listed in Table 2-7.

| Function Name | Test Target | Platform |
|---|---|---|

| | | |
|---|---|---|
| CLASSB_ADC_Test() | ADC Band-gap channel | M0/M23/M4/8051/ARM9 |
| CLASSB_MUX_Test() | ADC channel MUX | M0/M23/M4/8051/ARM9 |

Table 2-7 I/O Test Target on Different Platforms

## 2.3 Safety Functions in Startup Safety Check

When the system is powered on, test items can confirm whether the function of the MCU is correct through the test items for startup. When the test is completed, system can execute the application. If the MCU is detected abnormally, it will stop the system and show an error message.

| Test Items | Function Name | IEC60730-1 Annex H Items |
|---|---|---|
| CPU Register Test | CLASSB_Registers_Test() | 1.1 CPU Register Test |
| Program Counter Test | CLASSB_ProgramCounter_Test() | 1.3 Program Counter Test |

Table 2-8 Startup Test Items

## 2.4 Safety Functions in Run Time Safety Check

In the application execution, the test Items for run time detect the MCU function continuously. In order not to affect the efficiency of the application, these test items are part of whole function. It focuses on the key functions of the system. If the MCU is detected abnormally, it will stop the system and show an error message.

| Software Test | Function Name | IEC60730-1 Annex H Items |
|---|---|---|
| CPU Register Test | CLASSB_Registers_Test() | 1.1 CPU Register Test |
| Program Counter Test | CLASSB_ProgramCounter_Test() | 1.3 Program Counter Test |
| Stack Test | CLASSB_Stack_Test() | 1.3 Program Counter Test |
| Interrupt Test | CLASSB_Interrupt_Clock_Test() | 2.0 Interrupt Test |
| Clock Test | CLASSB_Interrupt_Clock_Test() | 2.0 Interrupt Test |
| Flash Test | CLASSB_Flash_Test() | 4.1 ROM Test |
| RAM Test | CLASSB_Ram_Test() | 4.2 RAM Test |
| ADC Test | CLASSB_ADC_Test() | 7.2 Analog I/O Test |
| Multiplexer Test | CLASSB_Multiplexter_Test() | 7.2 Analog I/O Test |

Table 2-9 Run Time Test Items

## 2.5 Safety State

There are safe states for every test function if the test result is failed. According to a specific error code, the safe state could be “Idle” or “Reset” state. In “Idle” state, the CPU will run a while(1) loop. In “Reset” state, the CPU will issue a system reset to reboot the system.

| Test Functions Fails | Safe State | Function Name | Error Code |
|---|---|---|---|
| CPU Register Test Fail | Idle/Reset | Startup Time / Run Time | 0x10 |
| Program Counter Test Fail | Idle/Reset | Startup Time / Run Time | 0x14 |
| Stack Test Fail | Idle/Reset | Run Time | 0x26 |
| Interrupt Test Fail | Idle/Reset | Run Time | 0x18 |
| Clock Test Fail | Idle/Reset | Startup Time | 0x1C |
| Flash Test Fail | Idle/Reset | Run Time | 0x20 |
| RAM MarchX Test Fail | Idle/Reset | Run Time | 0x25 |
| ADC Test Fail Run Time | Idle/Reset | Run Time | 0x50 |
| Multiplexer Test Fail Run Time | Idle/Reset | Run Time | 0x54 |

Table 2-10 The Safe Sate and Error Code for Failure Test

## 2.6 Data Storage Architecture

This STL uses the scatter file to arrange below items in ROM/RAM/DRAM memory space. The applications must refer the scatter file in example code to arrange them to achieve the STL purpose.

| Scatter File Allocate Address | Description | M0/M23/M4 | ARM9 | 8051 |
|---|---|---|---|---|
| PC_Test_1 Address | PC_Test_1 execute address | ROM | DRAM | ROM |
| PC_Test_2 Address | PC_Test_2 execute address | ROM | DRAM | ROM |
| Flash Pre-calculated Checksum | Pre-calculated Checksum located address | ROM | DRAM | ROM |
| Stack check Pattern_0 ~ Patter_3 | Location to store Pattern_0 ~ Pattern_3 | RAM | DRAM | RAM |
| Storage for Global Variable | Two Storage for global variable maintain | RAM | DRAM | RAM |
| Backup_Buffer Address | Backup buffer for Flash Test | RAM | DRAM | RAM |
| Guard band Address | (option) avoid to overwrite data by accidentally | RAM | DRAM | RAM |

Table 2-11 Scatter File Allocate Address for Different Platforms

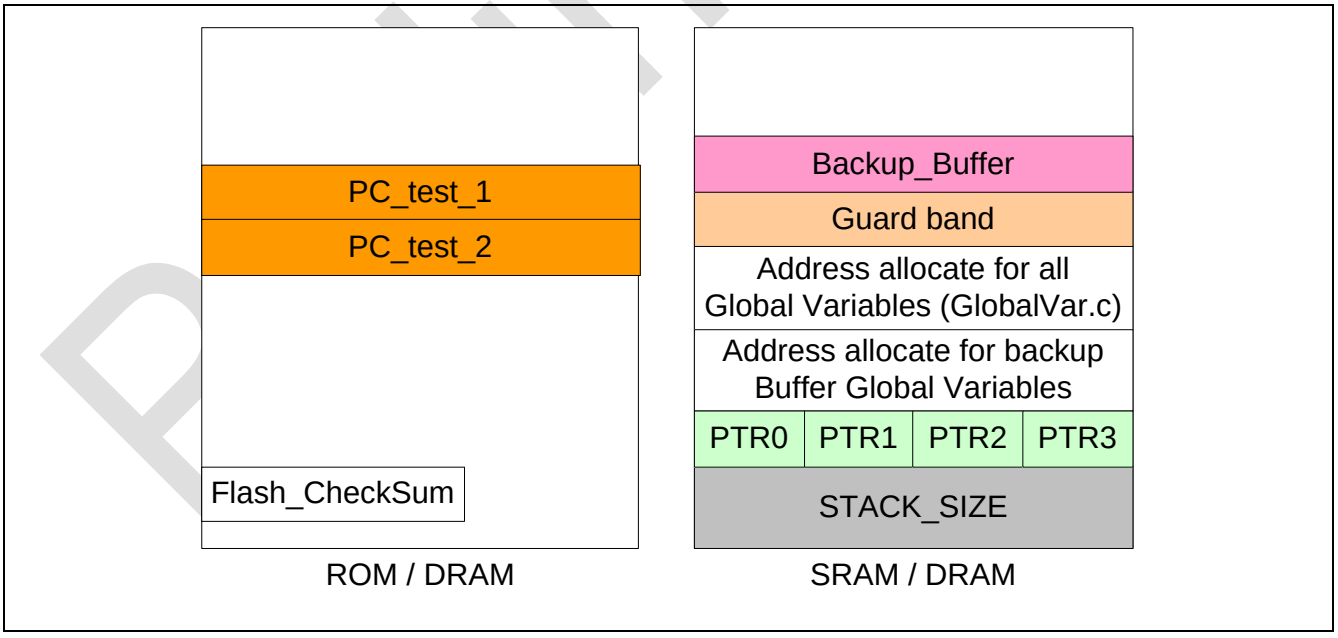



Figure 2-4 Scatter File Allocate Address Block Diagram

## 2.7 Timer0 and Timer1

Timer0 uses 100 Hz tick for example to generate an interrupt and increase the counter in ISR. Timer1 uses different clock source other than Timer0 and it can also generate its interrupt. The “u32HSFreq” is counting by Timer0, and its value will increase by 1 in the Timer0 ISR. The “u32LSFreq” is counting by Timer1, and its value will increase by 1 in the Timer1 ISR.

Runtime safety library will base on Timer0 Counter to find the time slot for safety functions to be invoked or not. Due to this Timer0 Counter is used to check whether each STL library function is in their checking time slot, it will reset each STL function counter at overflow. This can avoid the timer overflow and need a long time to wrap around the counter for next time test.

### 2.7.1 Function Error Detection Time

The software fault/error detection time listed in Table 2-12 are related to the execution periods of each test functions in “Run Time Check” which shows in Figure 2-5. In the beginning of the “Run Time Check” sequence, the “Execution Check” blocks will verify whether these test functions are going to be executed in this turn. If the timer0 Counter is greater than the execution counter of a function, then this function is going to be executed in this turn and the execution counter will increase by its execution cycle at the same time. The “Counter” could overflow after a long execution time. This should be checked and reset all counters in order to avoid that the “Counter” has been overflow and it won’t greater than “Execution Counter” for a long time. In this situation, all these test functions will not execute by their execution period any more.

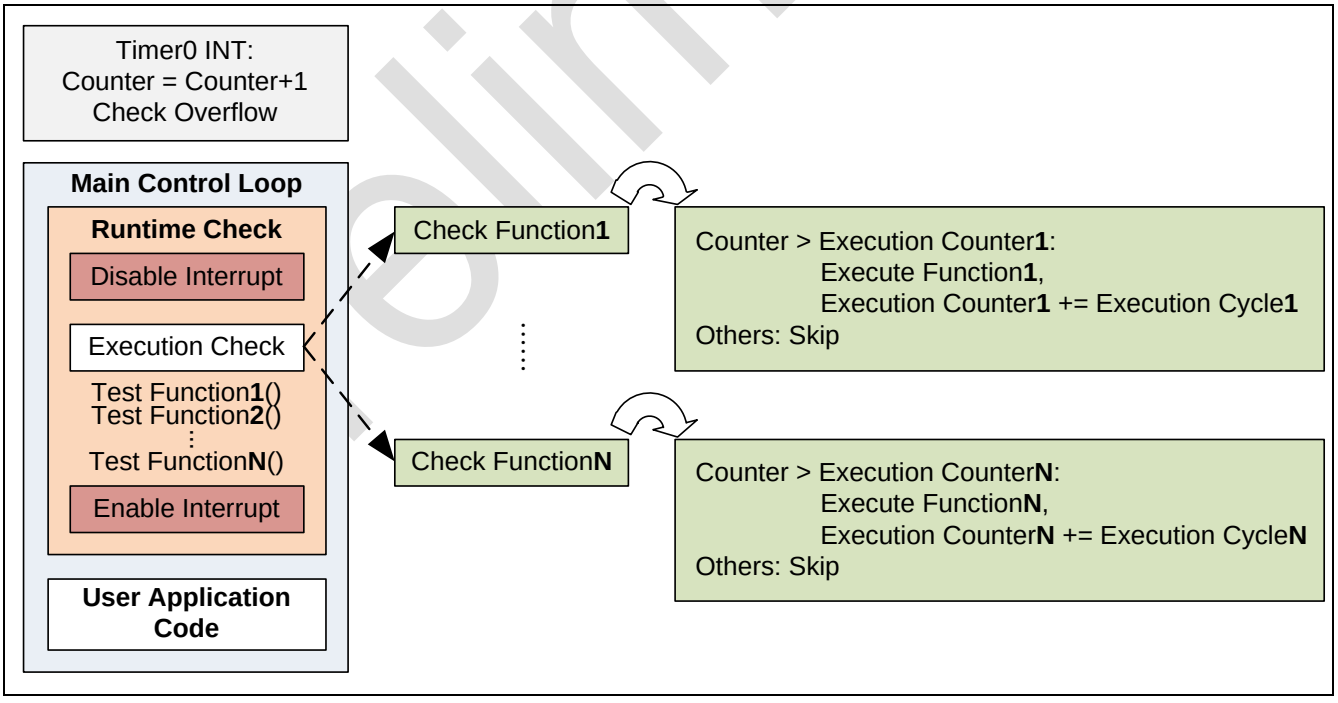



Figure 2-5 Software Fault/Error Detection Time

| Function Name | Error Detection Time |
|---|---|
| CLASSB_Registers_Test() | 2 Times the Execution Cycle of CPU |
| CLASSB_ProgramCounter_Test() | 2 Times the Execution Cycle of PC |
| CLASSB_Stack_Test() | 2 Times the Execution Cycle of Stack |
| CLASSB_Interrupt_Clock_Test() | 2 Times the high speed clock timer counting period |
| CLASSB_Interrupt_Clock_Test() | 2 Times the Execution Cycle of Clock Interrupt |
| CLASSB_Flash_Test() | 2 Times the (Execution Cycle of Flash * Flash Size /Check Size) |
| CLASSB_Ram_Test() | 2 Times the Execution Cycle of RAM |
| CLASSB_ADC_Test() | 2 Times the Execution Cycle of ADC |
| CLASSB_MUX_Test() | 2 Times the Execution Cycle of ADC Multiplexer |

Table 2-12 Software Fault/Error Detection Time in Run Time Check

### 2.7.2 Timer Deviation

Due to Timer0 & Timer1 time are based on different clock sources, it exists a ratio for these two timers. Except the ration between these two timers, there is a clock deviation between these two times that could result in a distortion from this ratio. Therefore, the STL set a tolerance value to compromise this clock deviation error. **Error! Reference source not found.** shows an example with ratio = 5 and tolerance = +/- 10%. When u32LSFreq=2, u32HSFreq in the range from 9 to 11 being considered as valid range.

If the interrupt cannot be generated to CPU, the counter will not increase. Then the ratio of Timer0 counter and Timer1 Counter will differ from the setting. And the error will be shown when running the interrupt and clock test.

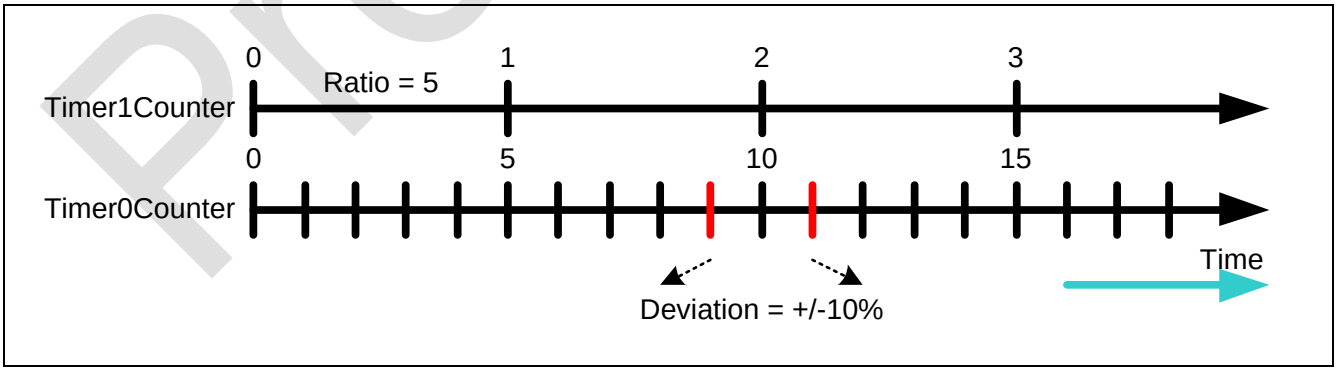



Figure 2-6 Clock Deviation and Ratio Setting Example

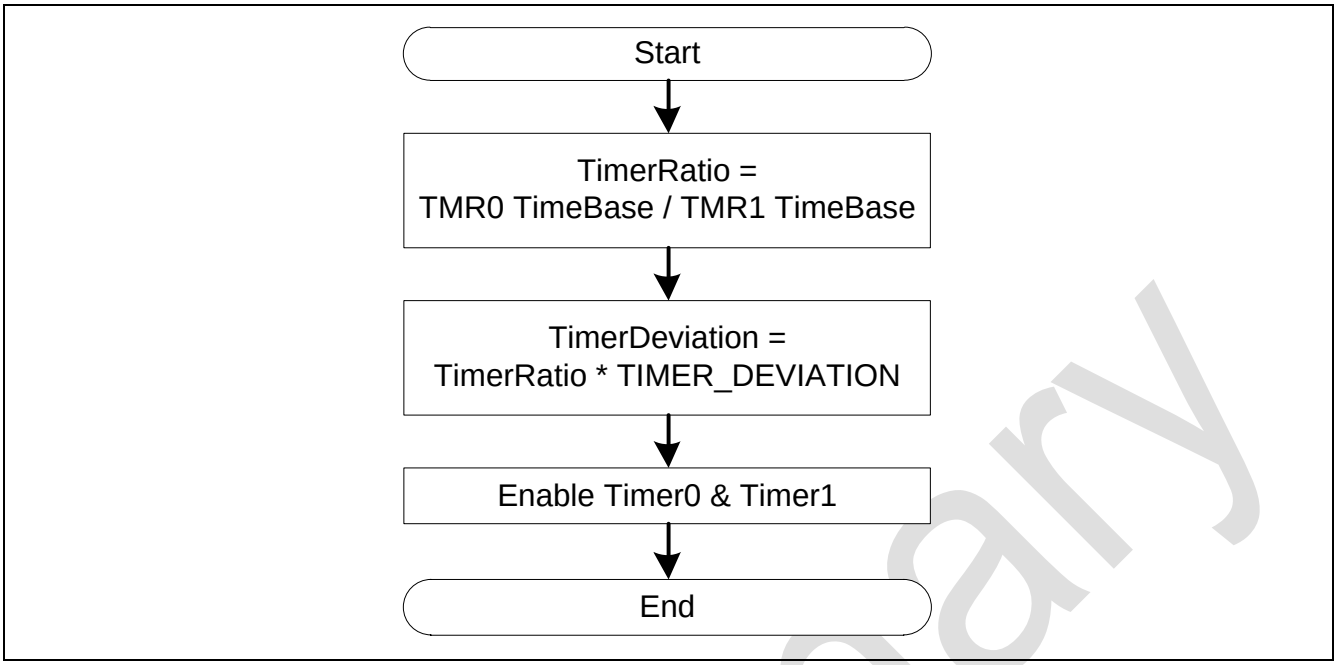



Figure 2-7 Clock Deviation and Ratio Setting Initial

### 2.7.3 Timer Counter Halt Situation

There are two run time tests must turn off interrupt before testing, “CPU_Reg_Test” and “RAM_Test”. If the interrupt has been disabled, the Timer Counter which is used in “Execution Check” function in previous section is also stopped. If the interrupt can be turned on before the next timer counter interrupt arrived, there should be no problem for a continuous counting sequence. otherwise, one or more counting on this Time Counter might be lost. Compare the execution time between these two functions and the Timer Counter. The test time is short enough and the counter will never have miss counting when interrupt is disabled during run time tests.

| Item | Max Test Time (μs) |
|---|---|
| CPU Register Test | 14 |
| RAM Test | 100 |
| **Item** | **Period (μs)** |
| Timer Counter | 10000 |

Table 2-13 Execution Time and the Timer Counter

## 2.8 Interfaces between Software and Hardware

The software here is related to CPU and storage (Flash, RAM) and hardware here is related to both digital and analog devices. For GPIO, PWM, communication port and debug Interface belong to digital devices; ADC, DAC, ACMP and OPA are analog devices. The interface between software and hardware is major by an internal “Bus Interfaces”, which services most command execution from software and data exchange between software and hardware. The “Register Bank” works over the bus interface, which can buffer the commands from software and update status from hardware at the same time. The other interface – “Interrupt” provides a quick response connection between software and hardware other than the “Bus Interface”, which usually service specific events trigger in a bi-direction way. “Clock” is a basic component for both software and hardware. There are always I/O Pins with these hardware devices, which are usually used to communicate between internal and external devices. ADC module has a multiplexer to switch from different sampling source. The sample source can come from either external pins or chip internal reference voltage.

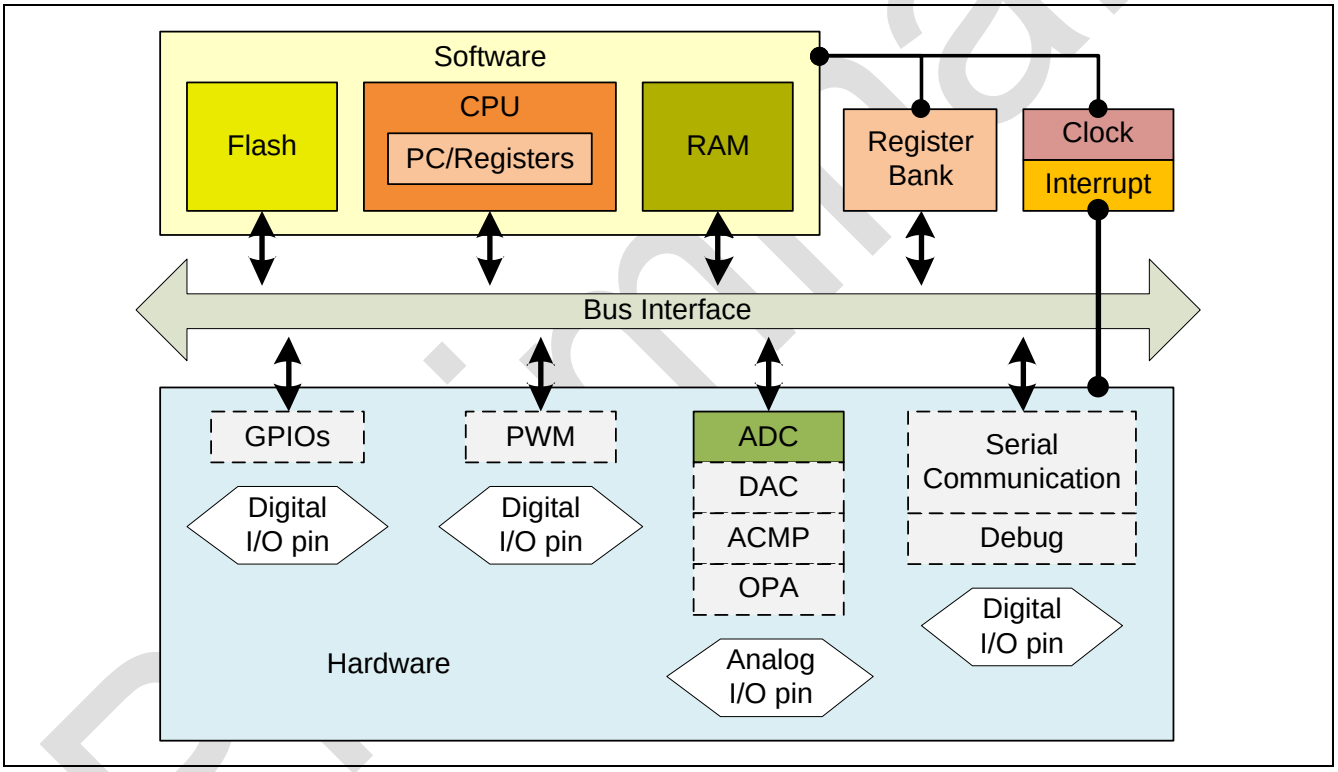



Figure 2-8 Software, Hardware and Interfaces

# 3 Safety Functions Description

## 3.1 CLASSB_STARTUP_TESTS

After system power on and finishing initialization, the module will be processed. This module will process two tests listed below. These modules will be descripted in the below sections.

- CLASSB_Registers_Test
- CLASSB_ProgramCounter_Test

If any one of the test is failed, the system will enter module CLASSB_SAFE_STATE (A).

If all tests in this module are passed, the system will then run the runtime test and user application.

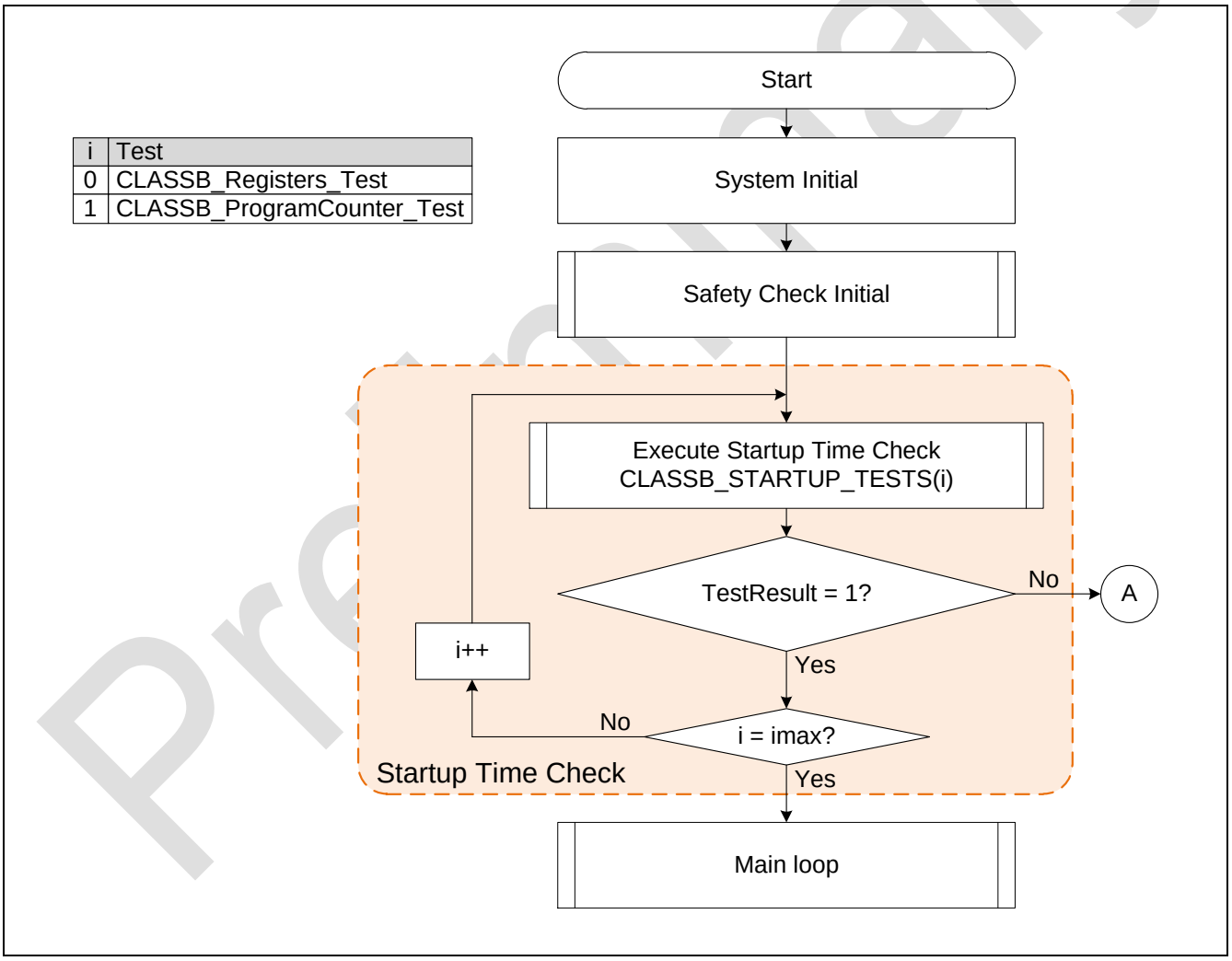



Figure 3-1 CLASSB_STARTUP_TESTS Block Diagram

## 3.2 CLASSB_RUNTIME_TESTS

If all start up tests has been passed, the system will then run the runtime tests in this module. This module will process eight tests listed below. These modules will be descripted in the below sections.

- CLASSB_Registers_Test
- CLASSB_ProgramCounter_Test
- CLASSB_Stack_Test
- CLASSB_RAM_Test
- CLASSB_Flash_Test
- CLASSB_Interrupt_Clock_Test
- CLASSB_ADC_Test
- CLASSB_MUX_Test

The module CLASSB_CHECK_RUNTIME_TESTS_EXECUTION will check the test flags depends on the counters increased in Timer. If the flag(s) is set to 1, the runtime test will be processed. If any one of the test is failed, the system will enter module CLASSB_SAFE_STATE (B). If all tests in this module are passed, the system will then keep running the runtime test and user application.

| j | TestFlag |
|---|---|
| 0 | REGISTER_TEST |
| 1 | PROGRAMCOUNTER_TEST |
| 2 | STACK_TEST |
| 3 | RAM_TEST |
| 4 | ROM_TEST |
| 5 | INTERRUPT_TEST |
| 6 | ADC_TEST |
| 7 | MUX_TEST |

| j | Test |
|---|---|
| 0 | CLASSB_Registers_Test |
| 1 | CLASSB_ProgramCounter_Test |
| 2 | CLASSB_Stack_Test |
| 3 | CLASSB_RAM_Test |
| 4 | CLASSB_Flash_Test |
| 5 | CLASSB_Interrupt_Clock_Test |
| 6 | CLASSB_ADC_Test |
| 7 | CLASSB_MUX_Test |

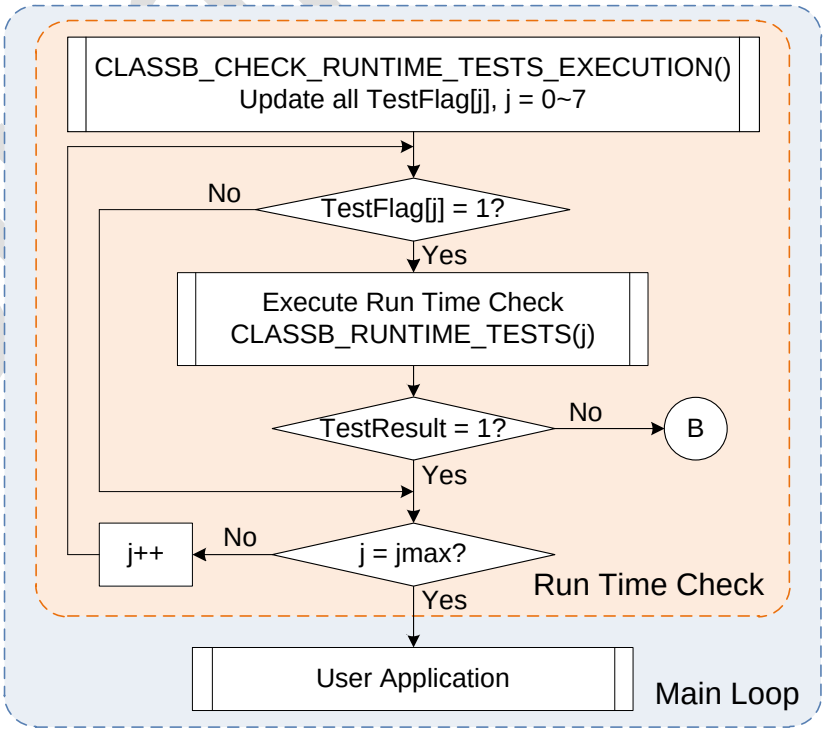
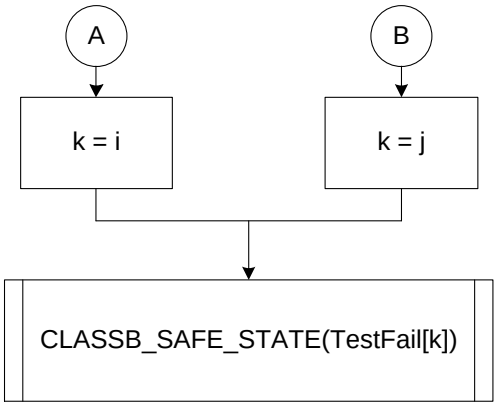



Figure 3-2 CLASSB_RUNTIME_TESTS Block Diagram

## 3.3 CLASSB_SAFE_STATE

If any test fails, the system will enter this module. Depends on the failure happened in which test module. These safe states are further to be defined by user either an idle state (CLASSB_IDLE_STATE) or a reset state (CLASSB_RESET_STATE). Under reset state, the system will do a system reset immediately. Under idle state, the system will enter a while(1) loop.

<table>
<tr><td>Format</td><td>uint8_t CLASSB_SAFE_STATE(uint8_t TestResult)</td></tr>
<tr><td>Arguments</td><td>uint8_t TestResult: Test result from test module.
<table>
<tr><th>TestResult</th></tr>
<tr><td>CPU_TEST_FAIL</td></tr>
<tr><td>RAM_TEST_FAIL</td></tr>
<tr><td>FLASH_TEST_FAIL</td></tr>
<tr><td>INTERRUPT_TEST_FAIL</td></tr>
<tr><td>ADC_TEST_FAIL</td></tr>
<tr><td>MUX_TEST_FAIL</td></tr>
</table>
<table>
<tr><th>Safe State</th></tr>
<tr><td>CLASSB_RESET_STATE</td></tr>
<tr><td>CLASSB_IDLE_STATE</td></tr>
</table></td></tr>
<tr><td>Global Values</td><td>None</td></tr>
<tr><td>Return Values</td><td>0</td></tr>
</table>




| i | Test | Test Result |
|---|---|---|
| 0 | CLASSB_Registers_Test | CPU_TEST_FAIL |
| 1 | CLASSB_ProgramCounter_Test | CPU_TEST_FAIL |
| | | |
| | | |

| j | Test | Test Result |
|---|---|---|
| 0 | CLASSB_Registers_Test | CPU_TEST_FAIL |
| 1 | CLASSB_ProgramCounter_Test | CPU_TEST_FAIL |
| 2 | CLASSB_Stack_Test | CPU_TEST_FAIL |
| 3 | CLASSB_RAM_Test | RAM_TEST_FAIL |
| 4 | CLASSB_Flash_Test | FLASH_TEST_FAIL |
| 5 | CLASSB_Interrupt_Test | INTERRUPT_TEST_FAIL |
| 6 | CLASSB_ADC_Test | ADC_TEST_FAIL |
| 7 | CLASSB_MUX_Test | MUX_TEST_FAIL |

Figure 3-3 CLASSB_SAFE_STATE Block Diagram

## 3.4 CLASSB_CHECK_RUNTIME_TESTS_EXECUTION

Every test module in CLASSB_RUNTIME_TESTS is a TIMER0 counter-based checking sequence for execution or not. This module checks the test cycle of each test module. If the test cycle satisfies the execution condition, it will set the execution flag to 1 and this test module will be executed in this turn. This flag will be clear after test module has finished it's testing. Please refer to section 2.7.1 for details.

## 3.5 CLASSB_REGISTERS_TEST

This test module will be executed in both startup time “CLASSB_STARTUP_TESTS” and runtime “CLASSB_RUNTIME_TESTS”.

| | |
|---|---|
| **Format** | uint8_t CLASSB_Registers_Test(uint8_t TestMode) |
| **Arguments** | uint8_t TestMode: Test result from test module.<br>STARTUP: under startup safety check, run the startup test module<br>RUNTIME: under run time safety check, run the runtime test module<br>The testing method under different test mode are the same. |
| **Global Values** | pattern1: 0x55555555<br>pattern2: 0xAAAAAAAA |
| **Return Values** | uint8_t TestResult<br>1: test pass<br>0: test fail |
| **Error Code** | CPU_TEST_FAIL |

### 3.5.1 Startup Safety Check

The software shall detect and respond to stuck-at error in CPU register by periodically checking the read-write specific test pattern. This module applied test patterns 0xAAAAAAAA and 0x55555555 regularly on CPU registers check the result by reading it back. If check fail, a TestResult:0 will be return with an error message and the test function is abort immediately. If test has passed, test module returns TestResult:1 instead.

The different MCUs have different numbers of registers:

For 8051 series, test module will verify the following registers.

- General purpose registers (R0 ~ R7)
- ACC

For M0 and M4 series, test module will verify the following registers.

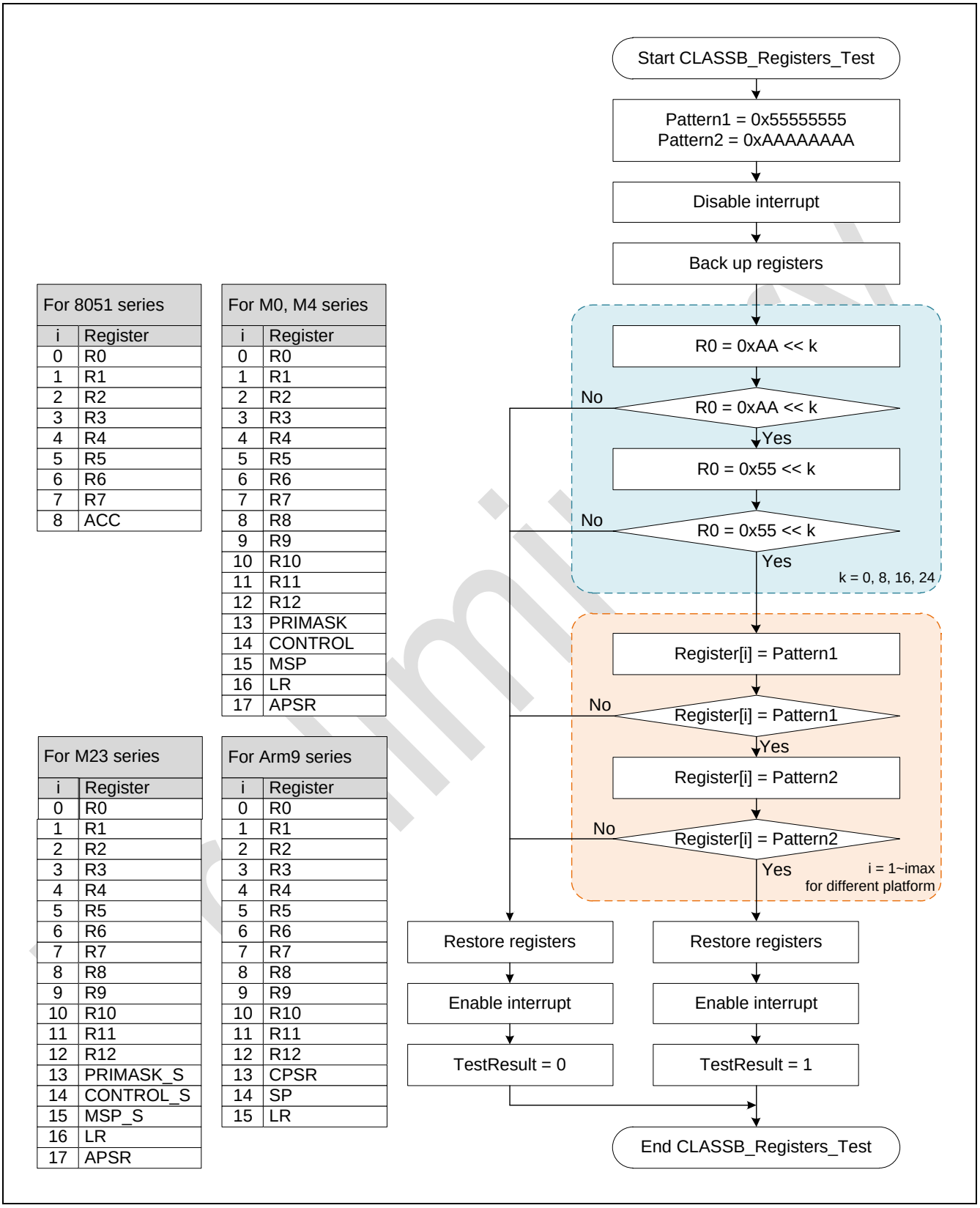
- General purpose registers (R0 ~ R12)
- PRIMASK register
- CONTROL register
- Main SP register (R13)
- LR register (R14)
- APSR register

For M23 series, test module will verify the following registers.

- General purpose registers (R0 ~ R12)
- PRIMASK register under Secure State
- CONTROL register under Secure State
- Main SP register (R13) under Secure State
- LR register (R14)
- APSR register

For Arm9 series, test module will verify the following registers.

- General purpose registers (R0 ~ R12)
- Current Process Status Register CPSR
- SP
- LR




For 8051 series

| i | Register |
|---|---|
| 0 | R0 |
| 1 | R1 |
| 2 | R2 |
| 3 | R3 |
| 4 | R4 |
| 5 | R5 |
| 6 | R6 |
| 7 | R7 |
| 8 | ACC |

For M0, M4 series

| i | Register |
|---|---|
| 0 | R0 |
| 1 | R1 |
| 2 | R2 |
| 3 | R3 |
| 4 | R4 |
| 5 | R5 |
| 6 | R6 |
| 7 | R7 |
| 8 | R8 |
| 9 | R9 |
| 10 | R10 |
| 11 | R11 |
| 12 | R12 |
| 13 | PRIMASK |
| 14 | CONTROL |
| 15 | MSP |
| 16 | LR |
| 17 | APSR |

For M23 series

| i | Register |
|---|---|
| 0 | R0 |
| 1 | R1 |
| 2 | R2 |
| 3 | R3 |
| 4 | R4 |
| 5 | R5 |
| 6 | R6 |
| 7 | R7 |
| 8 | R8 |
| 9 | R9 |
| 10 | R10 |
| 11 | R11 |
| 12 | R12 |
| 13 | PRIMASK_S |
| 14 | CONTROL_S |
| 15 | MSP_S |
| 16 | LR |
| 17 | APSR |

For Arm9 series

| i | Register |
|---|---|
| 0 | R0 |
| 1 | R1 |
| 2 | R2 |
| 3 | R3 |
| 4 | R4 |
| 5 | R5 |
| 6 | R6 |
| 7 | R7 |
| 8 | R8 |
| 9 | R9 |
| 10 | R10 |
| 11 | R11 |
| 12 | R12 |
| 13 | CPSR |
| 14 | SP |
| 15 | LR |

Figure 3-4 CLASSB_Registers_Test in Startup Safety Check Block Diagram

### 3.5.2 Runtime Safety Check

This module applied the same test patterns 0xAAAAAAAA and 0x55555555 as startup time test. The difference is some CPU registers are not tested during runtime.

For 8051 series, test module will verify the following registers.

- General purpose registers (R0 ~ R7)

For M0, M23, M4 and Arm9 series, test module will verify the following registers.

- General purpose registers (R0 ~ R12)

| For 8051 series | |
|---|---|
| i | Register |
| 0 | R0 |
| 1 | R1 |
| 2 | R2 |
| 3 | R3 |
| 4 | R4 |
| 5 | R5 |
| 6 | R6 |
| 7 | R7 |

| For M0, M23, M4 , and Arm9 series | |
|---|---|
| i | Register |
| 0 | R0 |
| 1 | R1 |
| 2 | R2 |
| 3 | R3 |
| 4 | R4 |
| 5 | R5 |
| 6 | R6 |
| 7 | R7 |
| 8 | R8 |
| 9 | R9 |
| 10 | R10 |
| 11 | R11 |
| 12 | R12 |

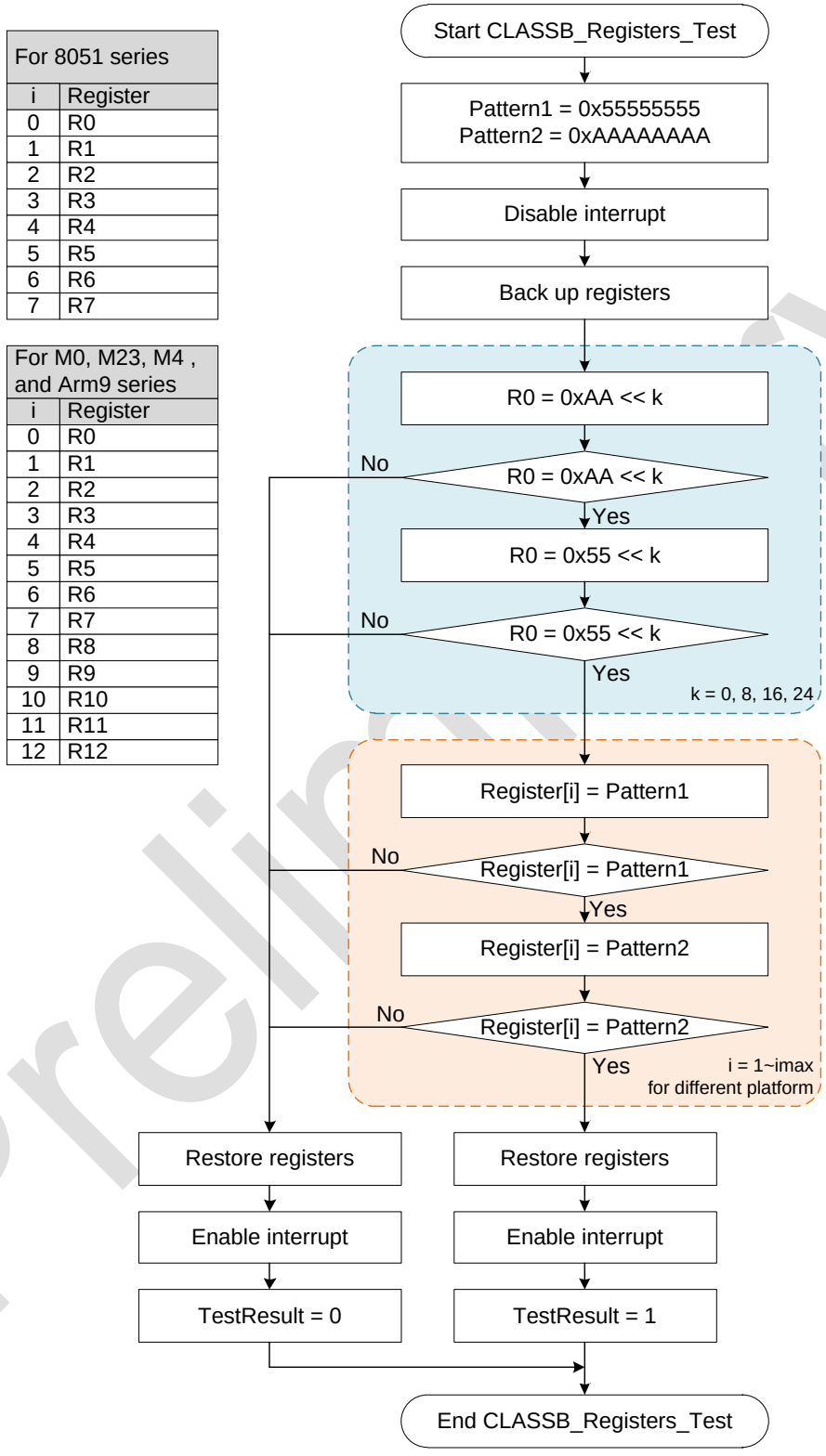



Figure 3-5 CLASSB_Registers_Test in Run Time Safety Check Block Diagram

## 3.6 CLASSB_PROGRAMCOUNTER_TSET

This test module will be executed in both startup time “CLASSB_STARTUP_TESTS” and runtime “CLASSB_RUNTIME_TESTS”. 8051, M0, M23, M4 and Arm9 series use the same following process.

The test module shall detect and respond whether MCU Program Counter can branch to the pre-defined address location or not. 8051, M0, M23, M4 and Arm9 series are all applying the same test process.

Test module predefine a function call address in a scatter file (this scatter file can help user to arrange the layout of code section). Two sections with different address, pc_test_1 and pc_test_2 are redefined in scatter file. These two address are complement, for example, pc_test_1 is put at address b1010101010100 and pc_test_2 is put at a bit toggle address b10101010101000. These addresses can be modified to fit the test environment by user. The pc_test_1 function returns the pre-defined value defined by test module and pc_test_2 returns the address of this function call.

| | |
|---|---|
| **Format** | uint8_t CLASSB_ProgramCounter_Test () |
| **Arguments** | None |
| **Global Values** | PC_RETURNVALUE: return value of sub-function pc_test_return1(). |
| **Return Values** | uint8_t TestResult<br>1: test pass<br>0: test fail |
| **Error Code** | CPU_TEST_FAIL |

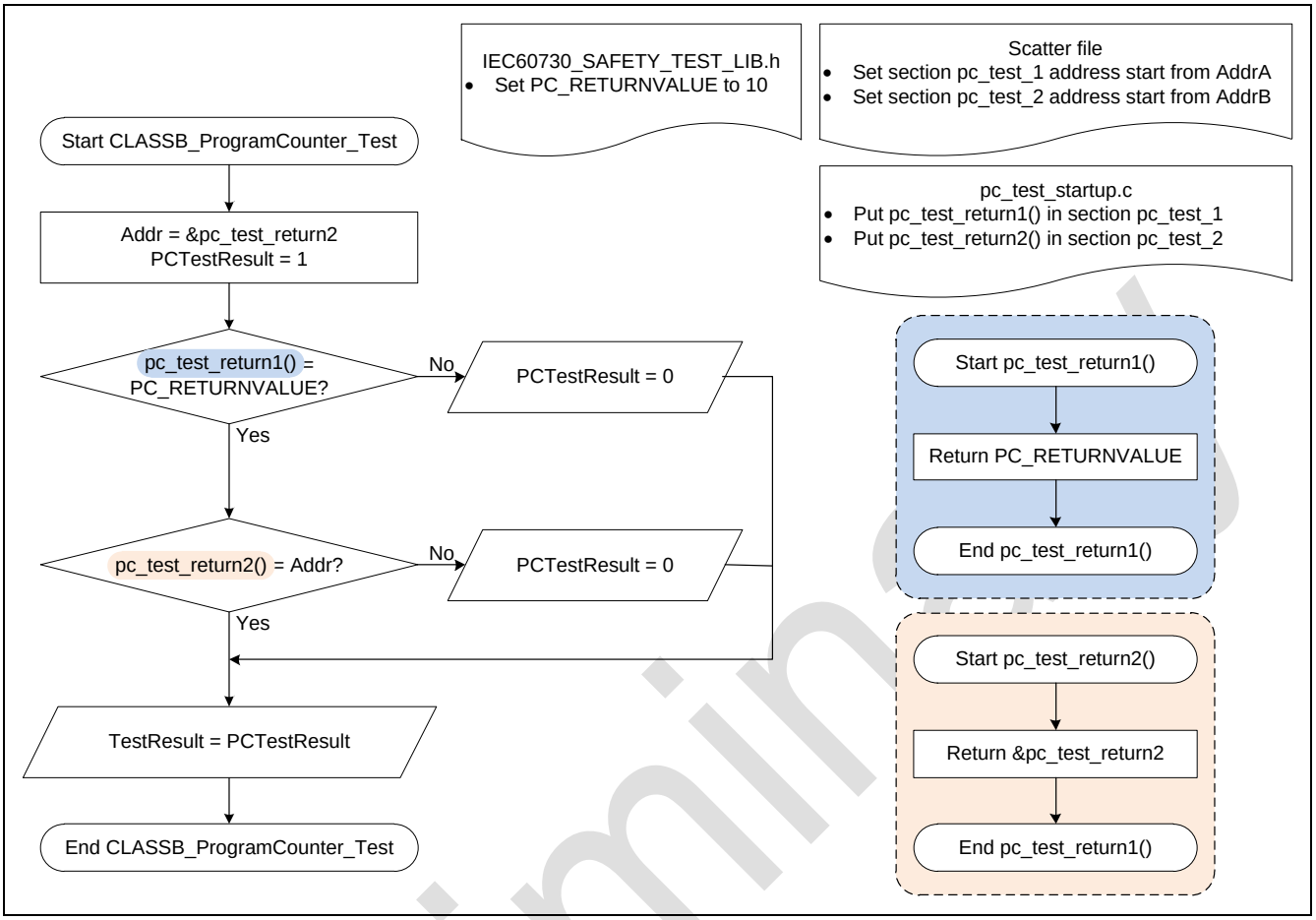



Figure 3-6 CLASSB_ProgramCounter_Test Block Diagram

## 3.7 CLASSB_STACK_TEST

This test module will be executed in runtime “CLASSB_RUNTIME_TESTS” only. 8051, M0, M23, M4 and Arm9 series use the same following process.

This module will check if the stack boundary is overflow or not. This module checks whether the specific pattern in the beginning of stack area is destroyed or not. If the pattern has not been changed, it means the Stack overrun condition never met. The specific pattern is defined and allocated by the scatter file. Please refer to section 2.6 for details.

| | |
|---|---|
| **Format** | uint8_t CLASSB_Stack_Test(uint32_t* pSTACK_TEST_PTRN) |
| **Arguments** | uint32_t* pSTACK_TEST_PTRN |
| **Global Values** | STACK_OVERRUN_PTRN0, STACK_OVERRUN_PTRN1, STACK_OVERRUN_PTRN2, STACK_OVERRUN_PTRN3: the patterns put in the memory on the top of the stack boundary. |
| **Return Values** | uint8_t TestResult<br>1: test pass<br>0: test fail |
| **Error Code** | CPU_TEST_FAIL |

## 3.8 CLASSB_RAM_TEST

This test module will be executed in runtime “CLASSB_RUNTIME_TESTS” only.

This module will check the data in the tested RAM area by default March C algorithm. For run time safety check, this module will test one area of SRAM at one time, and summary the test result after finish testing full area of SRAM.

| | |
|---|---|
| **Format** | uint8_t CLASSB_RAM_Test (uint8_t TestMode) |
| **Arguments** | uint8_t TestMode<br>RUNTIME: under run time safety check, run the runtime test module |
| **Global Values** | RAMStartAddr: The starting address of RAM.<br>RAMEndAddr: The end address of RAM<br>RAM_RUNTIME_TEST_LENGTH: The length of one runtime test area. |
| **Return Values** | uint8_t TestResult<br>1: test pass<br>0: test fail |
| **Error Code** | RAM_TEST_FAIL |

The SRAM layout is in two dimensions. All Bit-x where x is from 0 to 31 are located in one array with 4096 bits. Each row is comprised with 4 words (32 bits/word). When the SRAM bit contents are set as the following diagram, every bit value is different from its neighbor’s. According with this diagram, at least 32 bytes (8 words) are set as the length of one runtime test area.

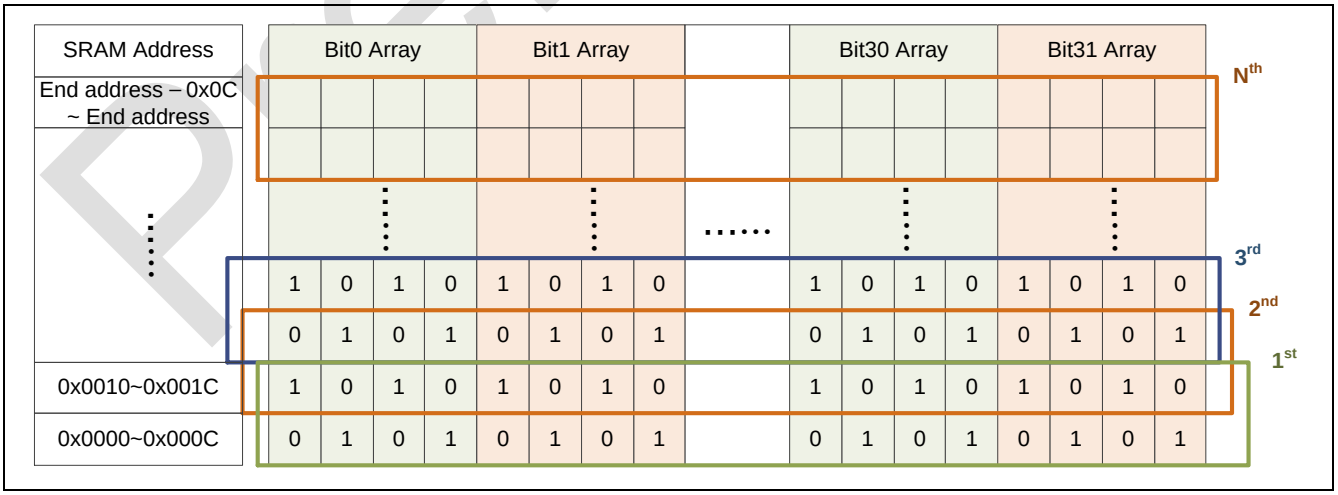



Figure 3-7 SRAM Layout

### 3.8.1 March C Algorithm

The test procedure of March C algorithm:

1. Clear all memory content to be zero.
2. Scan memory content from low to high address. For each memory location, check the read bit is 0 or not. If not, the test is failed otherwise write 1 back to the memory location.
3. Scan memory content from low to high address. For each memory location, check the read bit is 1 or not. If not, the test is failed otherwise write 0 back to the memory location
4. Scan memory content from high to low address. For each memory location, check the read bit is 0 or not. If not, the test is failed otherwise write 1 back to the memory location
5. Scan memory content from high to low address. For each memory location, check the read bit is 1 or not. If not, the test is failed otherwise write 0 back to the memory location
6. Check whether all bits are zero or not. The scanning direction can be either from high to low, or low to high address.

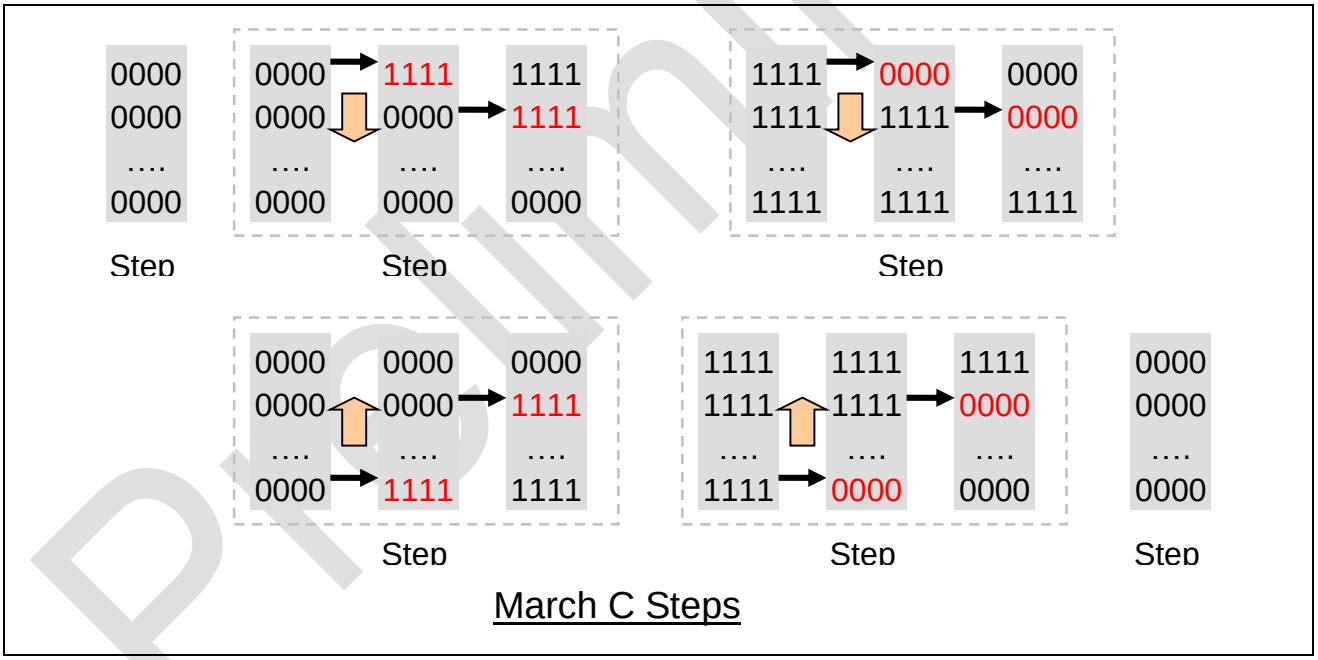



Figure 3-8 March C Algorithm

### 3.8.2 March X Algorithm

The test procedure of March X algorithm:

1. Clear all memory content to be zero.

2. Scan memory content from low to high address. For each memory location, check the read bit is 0 or not. If not, the test is failed otherwise write 1 back to the memory location.
3. Scan memory content from high to low address. For each memory location, check the read bit is 1 or not. If not, the test is failed otherwise write 0 back to the memory location
4. Check whether all bits are zero or not. The scanning direction can be either from high to low, or low to high address.

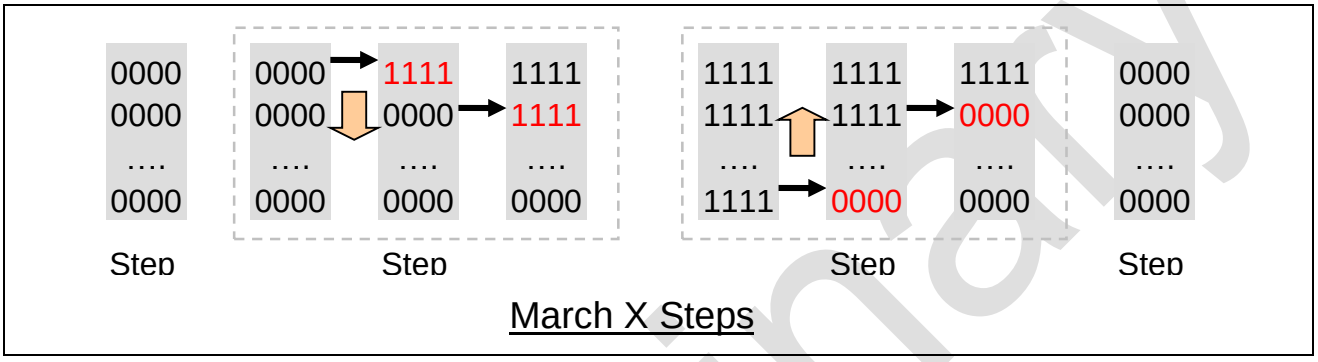



Figure 3-9 March X Algorithm

## 3.9 CLASSB_FLASH_TEST

This test module will be executed in runtime “CLASSB_RUNTIME_TESTS” only.

This module will check flash memory in the tested area by calculates the checksum and then compares the checksum with prior calculated value __Check_Sum last time to determine whether the test is passed or not.

| | |
|---|---|
| Format | uint8_t CLASSB_Flash_Test (uint8_t TestMode) |
| Arguments | uint8_t TestMode<br>RUNTIME: under run time safety check, run the runtime test module |
| Global Values | RomStartAddr: The starting address of ROM.<br>RomEndAddr: The end address of ROM<br>ROM_RUNTIME_TEST_LENGTH: The length of one runtime test area. |
| Return Values | uint8_t TestResult<br>1: test pass<br>0: test fail |
| Error Code | FLASH_TEST_FAIL |

For run time safety check in CALSSB_RUNTIME_TESTS, this module will test 32Byte flash section one by one at a time, and summary the test result until all finish has been tested over.

A pre-calculated flash content __Check_Sum value is store in flash by scatter file configuration. The __Check_Sum itself should not be included under pre-calculating flash content checksum procedure.

A scatter file is required to arrange the code layout in this testing. In this file description, there is an absolute address to define checksum value. The Flash program area is from address 0 to this checksum value address. User need to modify this address to fit the test environment.

The runtime flash test segment size is 8 bytes for 8051 series and 32 bytes for others. The checksums are calculated for every segments but only do the check at the end of the last flash segment has been done.

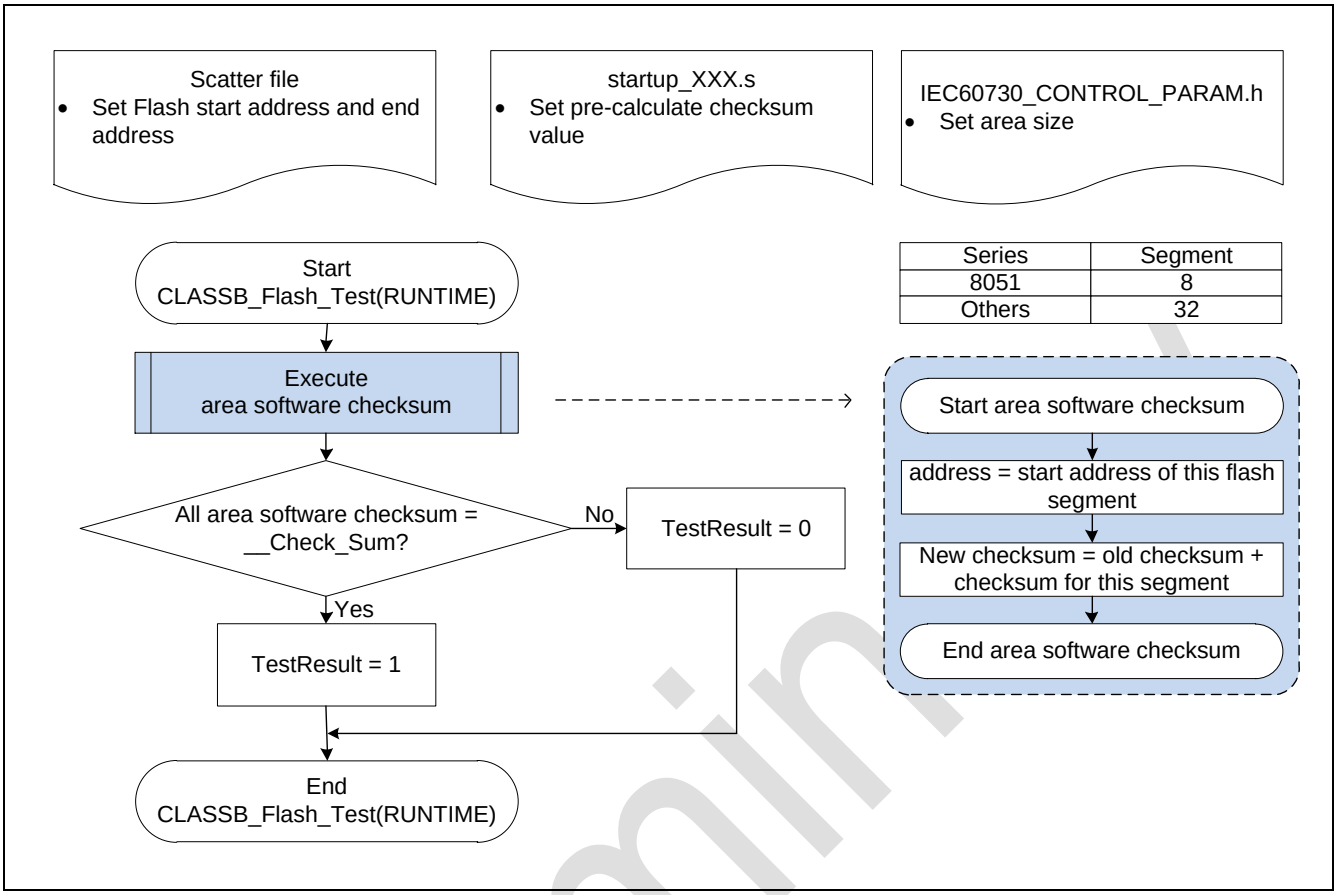



Figure 3-10 CLASSB_Flash_Test Block Diagram

## 3.10 CLASSB_INTERRUPT_CLOCK_TEST

This test module will be executed in runtime “CLASSB_RUNTIME_TESTS” only.

This module can test both clock and interrupt failure. This module uses two timers with different clock source and different frequency to count two counters. Timer0 use high speed clock source and timer1 use low speed clock source. Timer0 generate an interrupt with cycle A, and increase the first counter in timer0 ISR function. Timer1 generate an interrupt with cycle B, and increase the second counter in timer1 ISR function. If the ratio of two counters is not keep in a reasonable range, there should be a test failure for either clock or interrupt functions. Please refer to section 2.7 for details.

| | |
|---|---|
| **Format** | uint8_t CLASSB_Interrupt_Clock_Test() |
| **Arguments** | None |
| **Global Values** | HSCLOCK_FREQ: the first timer’s frequency<br>LSCLOCK_FREQ: the second timer’s frequency<br>CLOCK_DEVATION: the deviation of the timer |
| **Return Values** | uint8_t TestResult<br>1: test pass<br>0: test fail |
| **Error Code** | INTERRUPT_TEST_FAIL |

## 3.11 CLASSB_ADC_TEST

This test module will be executed in runtime “CLASSB_RUNTIME_TESTS” only.

In this module, it triggers ADC to sample the value of chip VREF. By ***directly*** comparing the sample value to the standard VREF constant, a test fail can be reported if there is a major difference other than the pre-defined threshold.

For 8051, M0, M23, M4 and Arm9 series, use the VREF to calculate the voltage of the MCU VBG. If the difference of these VBG value is larger than 10 %, the test is failed.

| | |
|---|---|
| **Format** | uint8_t CLASSB_ADC_Test() |
| **Arguments** | None |
| **Global Values** | |
| **Return Values** | uint8_t TestResult<br>1: test pass<br>0: test fail |
| **Error Code** | MUX_TEST_FAIL |

## 3.12 CLASSB_MUX_TEST

This test module will be executed in runtime “CLASSB_RUNTIME_TESTS” only.

In this module, it switches the channel of MUX to the VREF and use ADC to sample the value of the VREF like ADC test. This module will switch the channel of MUX to another one channel other than VREF.

| | |
|---|---|
| **Format** | uint8_t CLASSB_MUX_Test() |
| **Arguments** | None |
| **Global Values** | |
| **Return Values** | uint8_t TestResult<br>1: test pass<br>0: test fail |
| **Error Code** | MUX_TEST_FAIL |

# 4 Conclusion

When home appliances are under development, home appliances should follow the requirements specified in the IEC60730-1 standard for safety. Nuvoton provide an IEC60730-1 class B Software Test Library (STL) to implement MCU basic requirements specified in Annex H MCU part of the IEC60730-1 standard. It consists of several kinds of test items, including CPU register, Program Counter, WDT, Stack, Interrupt, RAM and Flash tests to detect the system fault/error. User can implement safety function based on the fault/error event to protect the system from dangers like fire hazard or motor rotor fault.

This STL collects common sets of tests dedicated mainly to generic blocks of M4 microcontroller family. User could take this application note to know the test items. According to the description of these test items, the user can include the STL package into a final project. In addition, the STL can be modified easily based on the application function of the product to achieve rapid development flow for IEC60730-1.

Finally, the user can implement a reliability and safety system compliant with IEC60730-1 safety standards.

# 5 Revision History

| Date | Revision | Description |
|---|---|---|
| 2021.04.12 | 1.00 | 1. Preliminary. |

## Important Notice

**Nuvoton 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, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.**

**All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.**