Application Note

TXZ+ Microcontroller Functional Safety STL

Table of Contents

1. Introduction

1.1. Purpose

This document describes the technical details of the TXZ+microcontroller functional safety STL for software developers.

1.2. Definitions, acronyms, and abbreviations

Terms/Abbreviation	Definition
User	Customers using Toshiba'smicrocontrollers
STL	Self-Test Library
STK	Starter Kit(Evaluation Board)
IEC60730	International Standards for Ensuring the Safety of Household Products
APN	Application Note
RM	Reference manual
TD	Datasheet
ES	Engineering Sample

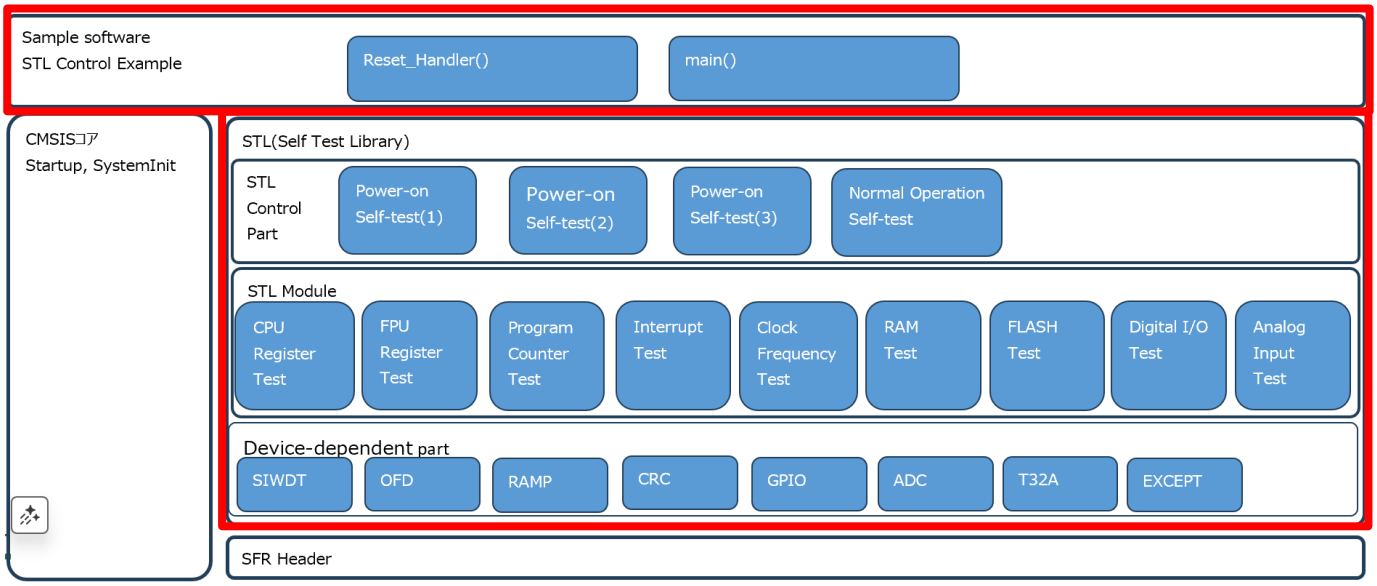
1.3. References

Reference Manual	IP Symbol
IEC60730 International Standard	-
Arm Cortex-M4 Processor Technical Reference Manual	-
TXZ+FamilyTMPM4K Group (2) datasheet	-
CRC Calculation Circuit	CRC-A
Oscillation Frequency Detection	OFD-A
12-bit Analog to Digital Converter	ADC-F
Clock Control and Operation Mode (TMPM4K Group (2))	CG-M4K (2)-E
Exception (TMPM4K Group (2))	EXCEPT-M4K (2)
Flash Memory	FLASH10MUD32-A
Product Information (TMPM4K Group (2))	PINFO-M4K (2)
Input/Output Ports (TMPM4K Group (2))	PORT-M4K (2)
32-bit Timer Event Counter	T32A-C
RAM Parity	RAMP-B
TXZ+FamilyTMPM4M Group (1) datasheet	-
Clock Control and Operation Mode (TMPM4M Group (1))	CG-M4M (1)-E
Exception (TMPM4M Group (1))	EXCEPT-M4M (1)
Flash Memory	FLASH512UD32-B
Product Information (TMPM4M Group (1))	PINFO-M4M (1)
Input/Output Ports (TMPM4M Group (1))	PORT-M4M (1)

2. Overall description

2.1. Specification overview

- This functional safety STL consists of a self-test library that can be used for Class B devices according to the European safety standard IEC/UL60730 Annex H, and sample software that utilizes the self-test library.
- This functional safety STL provides sample software that can be used to verify operation on evaluation boards for the TXZ+ family M4K(2) group, M4M group.



Note: The red frame indicates the scope of this functional safety STL.

2.2. STL functions overview

- The following Functional Safety STL features are provided.

Name	Test Content	IEC 60730-1 Category
CPU Register Test	Write the checker pattern 0x55555555 and 0xaaaaaaaa to all readable and writable internal registers. And verify that they can be read back correctly.	1.1 CPU Registers
FPU Register Test	Write the checker pattern 0x55555555 and 0xaaaaaaaa to all readable and writable internal registers. And verify that they can be read back correctly.	1.1 CPU Registers
CPU Program Counter Test	After executing the jump instruction, verify that the value of the program counter (PC) matches the expected value at the time of program description (build time). Prepare 10 different patterns for the jump destination address, and ensure that the program returns to the caller after all patterns are executed successfully.	1.2 CPU Program Counter
Interrupt Test	Using timer interrupts, verify that interrupt processing occurs at the expected frequency within a certain time period, and the interrupt processing completes successfully. If no interrupts occur or interrupts occur significantly more frequently than designed, it should be considered an error.	2. Interrupt handling and execution
Clock Frequency Test	Using the frequency detection circuit (OFD) and the reference clock input (IHOSC2), check for clock frequency abnormalities in the system clock(f_c) or the external high-speed oscillator(f_{EHOSC}) over a certain period of time. After the check period has elapsed, return whether a clock abnormality has occurred using the OFD.	3. Clock
RAM Test	Verify that the checker patterns 0x55555555 and 0xaaaaaaaa can be correctly written to and read from the same memory address, and confirm using the March C algorithm. Execute this for the specified memory range. If the device has a RAMP function, use the RAMP function for verification.	4.2 Variable Memory
FLASH Test	Calculate the CRC32 value for the specified address range and verify that the FLASH write has not been read back as a different value. Compare it with the CRC32 value that was pre-calculated at the time of program build. If the device does not have a CRC function, calculate it using software CRC.	4.1 Invariable memory
Digital Input/Output Test	Enable both input and output for I/O port and test by inputting the output state of the same port. Check whether the input terminal is Hi when Hi is output, and whether the input terminal is Lo when Lo is output.	7.1 Digital I/O
Analog Input Test	Generate three types of input voltages (VREFL/VREFH/REGOUT1) for input verification and test the values obtained through AD conversion.	7.2 Analog I/O

2.3. Development environment

- The development languages are C and assembly language.
- The build environments are IAR EWARM, ARM KEIL-MDK.
- The development environments are as following.

Resource Name	Purpose
PC	OS is Windows11.
IAR EWARM 9.50.2	IAR's IDE (Integrated Development Environment)
Arm® Keil™ MDK 5.40.0	ARM's IDE (Integrated Development Environment)

3. Specific requirements

3.1. Configurations

3.1.1. Tree Structure

TXZplusSTL ├─Examples │ └─src │ └─STL_app	STL Sample software
├─MCU │ ├─CMSIS │ ├─CMSIS_M4K │ │ └─startup │ ├─CMSIS_M4M │ │ └─startup │ └─Driver │ ├─inc │ └─src	MCU Specific definition Device-dependent part
├─PJ │ ├─TMPM4K │ │ ├─arm │ │ └─iar │ └─TMPM4M │ ├─arm │ └─iar	Project folder
└─SelfTestLibrary ├─inc └─src	Self-test library main body

3.1.2. File List

File List for STL Sample software, Self-test library, Device-dependent part.

File Name	Description
main.c	Sample software for STL
SelfTest_cpu_register.s	CPU Register Test
SelfTest_fpu_register.s	FPU Register Test
SelfTest_fpu_control.s	FPU Control Register Test
SelfTest_cpu_pc.c	Program Counter Test
SelfTest_interrupt.c	Interrupt Test
SelfTest_clock.c	Clock Frequency Test
SelfTest_ram_cheker.c	RAM Test Checker Pattern
SelfTest_ram_march_c.c	RAM Test MARCH C
SelfTest_ram_ramp.c	RAM Test RAM Parity
SelfTest_flash_crc_hw.c	FLASH Test Hardware CRC
SelfTest_flash_crc_sw.c	FLASH Test Software CRC
SelfTest_digital_io.c	Digital Input/Output Test
SelfTest_analog_in.c	Analog Input Test
SelfTest_adc.c	Device-dependent part A/D Converter
SelfTest_cg.c	Device-dependent part Clock Generator
SelfTest_gpio.c	Device-dependent part Port
SelfTest_except.c	Device-dependent part Interrupt
SelfTest_crc.c	Device-dependent part CRC
SelfTest_ofd.c	Device-dependent part OFD
SelfTest_ramp.c	Device-dependent part RAM Parity

3.1.3. API List

API List for Self-test library.

File Name	API Name	Function
SelfTest_cpu_register. s	SelfTest_CPU_Register	CPU Register Test
SelfTest_fpu_register. s	SelfTest_FPU_Register	FPU Register Test
SelfTest_fpu_control. s	SelfTest_FPU_Control	FPU Control Register Test
SelfTest_cpu_pc.c	SelfTest_CPU_ProgramCounter	Program Counter Test
SelfTest_interrupt.c	SelfTest_Interrupt	Interrupt Test
SelfTest_clock.c	SelfTest_Clock	Clock Frequency Test
SelfTest_ram_checker.c	SelfTest_RAM_CheckerPattern	RAM Test Checker Pattern
SelfTest_ram_march_c.c	SelfTest_RAM_March_C	RAM Test March C
SelfTest_ram_ramp.c	SelfTest_RAM_ramp_init	RAM Test RAM Parity Function Initialization
	SelfTest_RAM_ramp	RAM Test RAM Parity
	SelfTest_RAM_ramp_handler	RAM Test RAM Parity Function Interrupt Handler
SelfTest_flash_crc_hw.c	SelfTest_FLASH_CRC_HW	FLASH Test Hardware CRC
SelfTest_flash_crc_sw.c	SelfTest_FLASH_CRC_SW	FLASH Test Software CRC
SelfTest_digital_io.c	SelfTest_DigitalIO	Digital Input/Output Test
SelfTest_analog_in.c	SelfTest_AnalogIn	Analog Input Test

3.2. Function details

TXZ+ Microcontroller Functional Safety STL provides the following features.

3.2.1. CPU Register Test

Perform “Stuck at” Error Check (Note1) on the following CPU Registers.
Write the Checker Patterns 0x55555555 and 0xAAAAAAAA to the CPU Registers and Verify Correct Readout.

- General purpose registers (R0 – R12)
- Stack pointer (R13 (Main SP and Process SP) (Note2))
- Link register (R14 (LR))
- Control registers (CPSR (Upper 5 Bits Only))

CPU Register Test is performed sequentially. If an error is detected in a register, the test is interrupted at that point and an error is returned. At the same time, a bitmask value representing the register where the error occurred is returned.

- This API is intended to be called in privileged state mode (not user state).
- This API must be called in an interrupt-disabled state. Registers that are non-destructive during function calls in C language are preserved before and after the test.
- This API does not use interrupts.

Caution: Since executing the program always uses CPU registers, self-verification cannot completely detect defects.

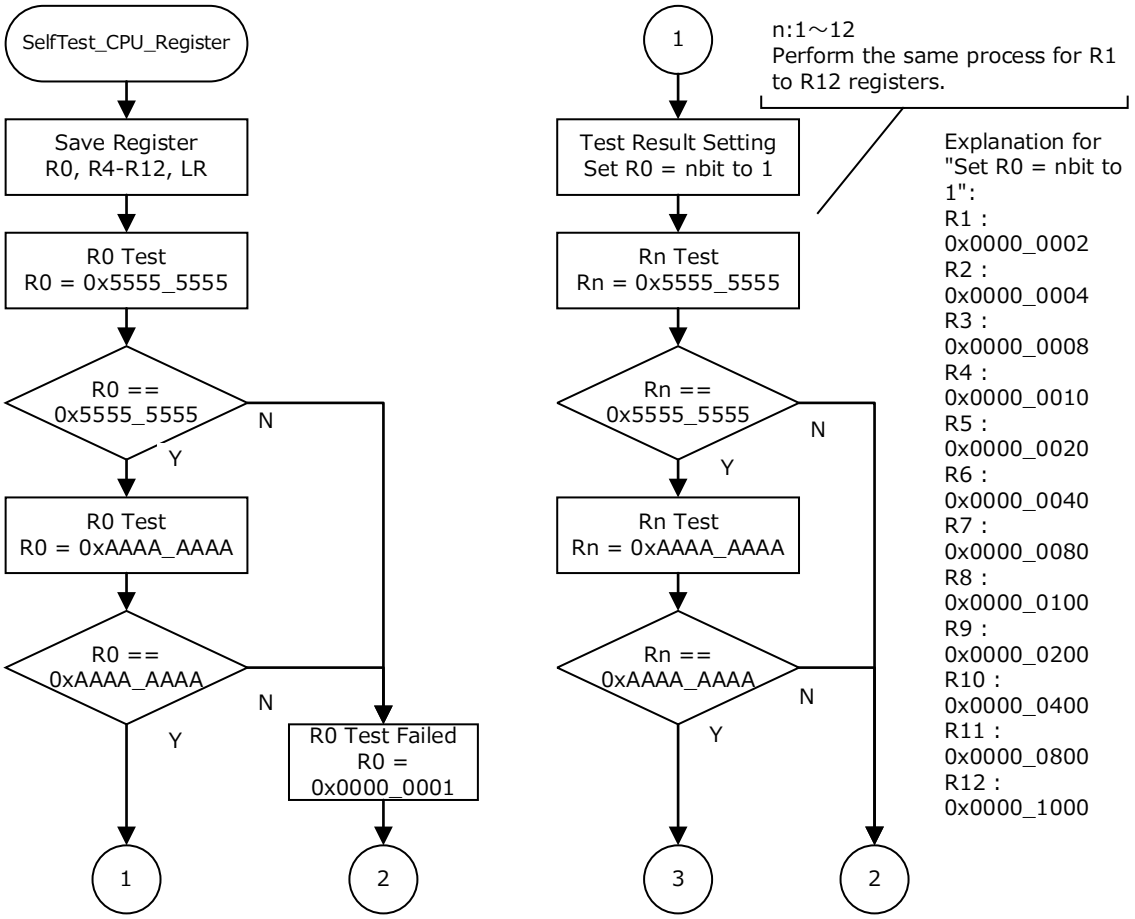
Note1: "Stuck at" check is a test that detects abnormalities where a bit is fixed at 0 or 1.
Note2: The lower 2 bits of the SP register are fixed at 0. These fixed bits are not tested.

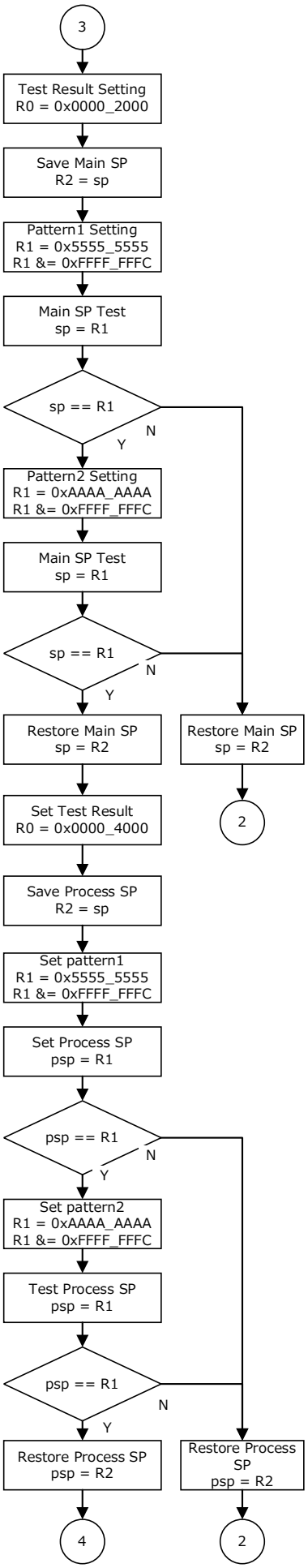
Source files: SelfTest_cpu_register.s
Header files: SelfTest_cpu_register.h

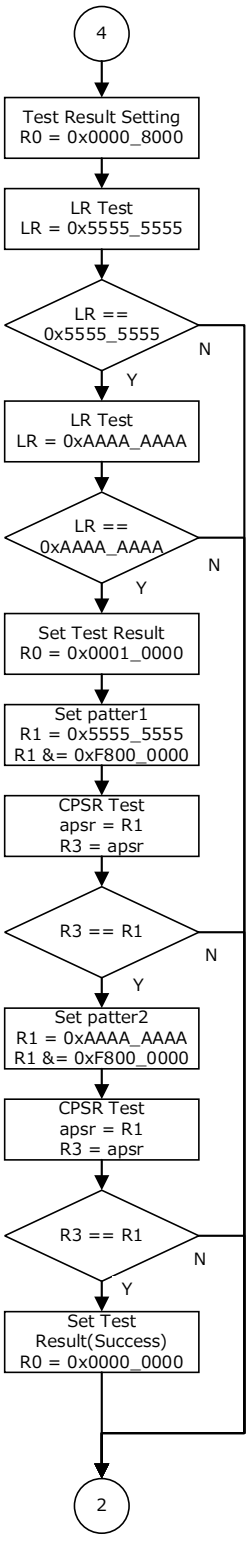
Function name
bool SelfTest_CPU_Register(uint32_t *result_word)
Explanation
Save the registers. General purpose registers execute tests on R0~R12. Write 0x5555_5555 to R0 Compare R0 with 0x5555_5555. If there is a mismatch, write 1 to bit0 of R0 and jump to error handling. Write 0xAAAA_AAAA to R0 Compare R0 with 0xAAAA_AAAA. If there is a mismatch, write 1 to bit0 of R0 and jump to error handling. Write 0x5555_5555 to R1 Compare R1 with 0x5555_5555. If there is a mismatch, write 1 to bit1 of R0 and jump to error handling. Write 0xAAAA_AAAA to R1 Compare R1 with 0xAAAA_AAAA. If there is a mismatch, write 1 to bit1 of R0 and jump to error handling. Subsequently, execute the same process for R2, R3, R4... R12. If there is a mismatch, write 1 to the corresponding bit in the output parameters of R0 and jump to error handling. Execute the test for Main SP (MSP) Save the MSP. Write 0x5555_5554 to the MSP (lower 2 bits are set to 0). Compare MSP with 0x5555_5554. If there is a mismatch, write 1 to bit13 of R0, restore MSP, and jump to error handling. Write 0xAAAA_AAA8 to the MSP (lower 2 bits are set to 0). Compare MSP with 0xAAAA_AAA8. If there is a mismatch, write 1 to bit13 of R0, restore MSP, and jump to error handling. If the test is successful, restore MSP and execute the test for Process SP. Execute the Process SP (PSP) test. Save the PSP. Write 0x5555_5554 to the PSP (the lower 2 bits are set to 0). Compare the PSP with 0x5555_5554, and if there is a mismatch, write 1 to bit14 of R0, restore the PSP, and jump to error handling. Write 0xAAAA_AAA8 to the PSP (the lower 2 bits are set to 0). Compare the PSP with 0xAAAA_AAA8, and if there is a mismatch, write 1 to bit14 of R0, restore the PSP, and jump to error handling. If the test is successful, restore the PSP and execute the Link register test. Execute the Link register (LR) test. Write 0x5555_5555 to the LR. Compare the LR with 0x5555_5555, and if there is a mismatch, write 1 to bit15 of R0 and jump to error handling. Write 0xAAAA_AAAA to the LR. Compare the LR with 0xAAAA_AAAA, and if there is a mismatch, write 1 to bit15 of R0 and jump to error handling. Execute the CPSR test. Write 0x5555_5555 to the LR. Compare the LR with 0x5555_5555, and if there is a mismatch, write 1 to bit15 of R0 and jump to error handling. Write 0xAAAA_AAAA to the LR. Compare the LR with 0xAAAA_AAAA, and if there is a mismatch, write 1 to bit15 of R0 and jump to error handling. Execute the CPSR test. Write 0x5000_0000 to the APSR (the upper 5 bits are CPSR, so the other bits are set to 0). Compare the APSR with 0x5000_0000, and if there is a mismatch, write 1 to bit16 of R0 and jump to error handling. Write 0xA800_0000 to the APSR (the upper 5 bits are CPSR, so the other bits are set to 0). Compare the APSR with 0xA800_0000, and if there is a mismatch, write 1 to bit16 of R0 and jump to error handling. Restore the saved registers. Set R0 to Output Parameters.

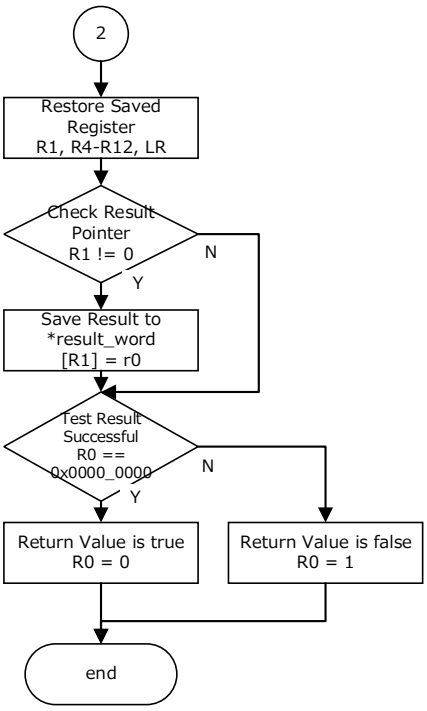
If the Argument Output Parameters is not set, the Return Value (R0) will be 0x01. When the test is complete, the Return Value (R0) will be 0x00.	
Input Parameters	
None	
Output Parameters	
uint32_t *result_word	Memory buffer to store the results bit0: R0 Test Failed - bit12: R12 Test Failed bit13: Main SP Test Failed bit14: Process SP Test Failed bit15: LR Test Failed bit16: CPSR Test Failed
Return Value	
Bool	Test Result (true: Success, false: Failed)

- Test Flowchart




3
Test Result Setting
R0 = 0x0000_2000
Save Main SP
R2 = sp
Pattern1 Setting
R1 = 0x5555_5555
R1 &= 0xFFFF_FFFC
Main SP Test
sp = R1
sp == R1
N
Y
Pattern2 Setting
R1 = 0xAAAA_AAAA
R1 &= 0xFFFF_FFFC
Main SP Test
sp = R1
sp == R1
N
Y
Restore Main SP
sp = R2
Restore Main SP
sp = R2
2
Set Test Result
R0 = 0x0000_4000
Save Process SP
R2 = sp
Set pattern1
R1 = 0x5555_5555
R1 &= 0xFFFF_FFFC
Set Process SP
psp = R1
psp == R1
N
Y
Set pattern2
R1 = 0xAAAA_AAAA
R1 &= 0xFFFF_FFFC
Test Process SP
psp = R1
psp == R1
N
Y
Restore Process SP
psp = R2
Restore Process SP
psp = R2
4
2


4
Test Result Setting
R0 = 0x0000_8000
LR Test
LR = 0x5555_5555
LR ==
0x5555_5555
N
Y
LR Test
LR = 0xAAAA_AAAA
LR ==
0xAAAA_AAAA
N
Y
Set Test Result
R0 = 0x0001_0000
Set patter1
R1 = 0x5555_5555
R1 &= 0xF800_0000
CPSR Test
apsr = R1
R3 = apsr
R3 == R1
N
Y
Set patter2
R1 = 0xAAAA_AAAA
R1 &= 0xF800_0000
CPSR Test
apsr = R1
R3 = apsr
R3 == R1
N
Y
Set Test
Result(Success)
R0 = 0x0000_0000
2


2
Restore Saved
Register
R1, R4-R12, LR
Check Result
Pointer
R1 != 0
N
Y
Save Result to
*result_word
[R1] = r0
Test Result
Successful
R0 ==
0x0000_0000
N
Y
Return Value is true
R0 = 0
Return Value is false
R0 = 1
end

3.2.2. FPU Register Test

Perform the "Stuck at" error check on the following FPU registers.
Write the checker patterns 0x55555555 and 0xaaaaaaaa, and verify that they can be read correctly.

- FPU Extended Registers (S0~S31)
- FPU Control registers (CPACR, FPCCR, FPCAR, FPSCR, FPDSCR)

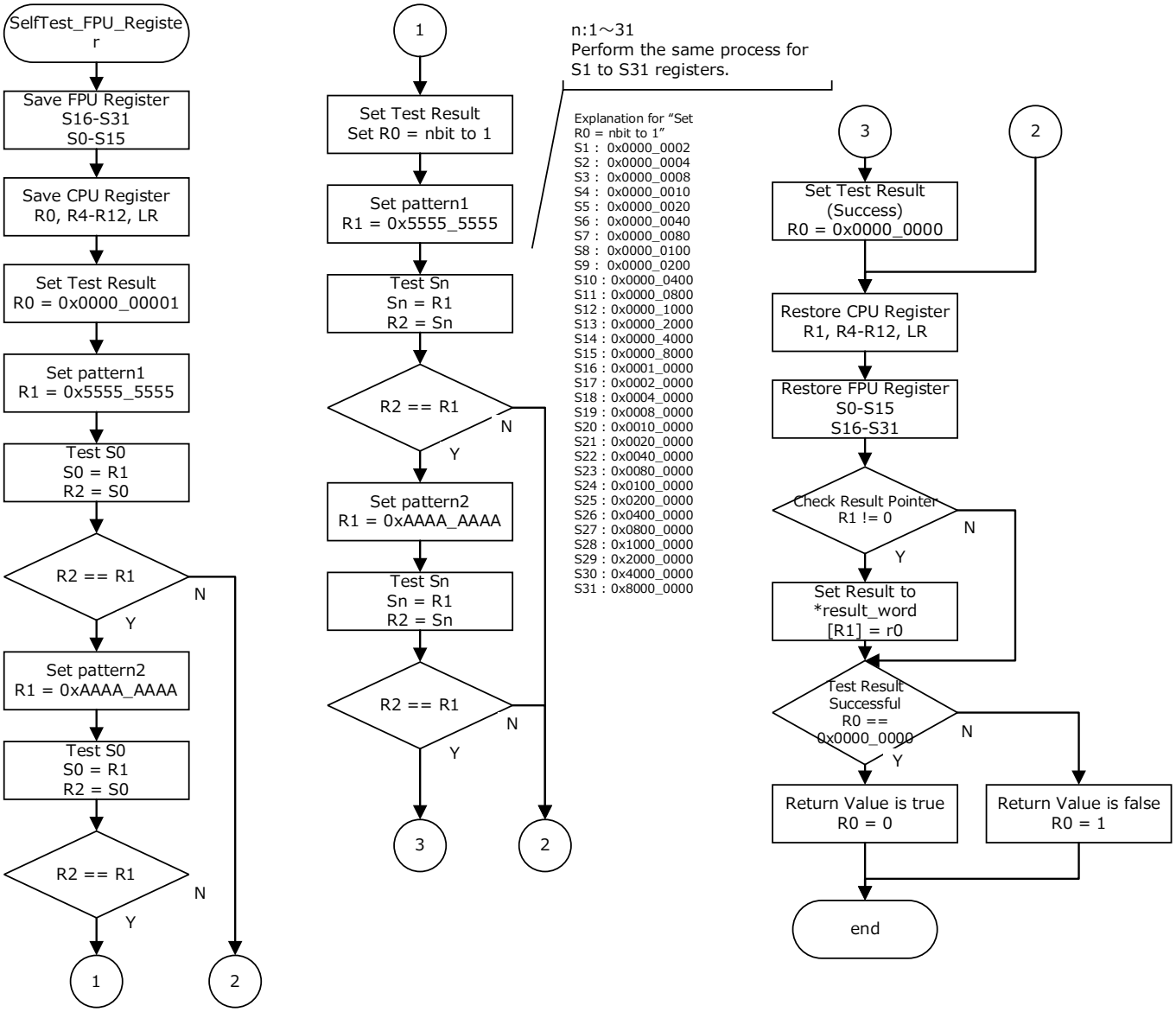
The FPU Register Test is performed sequentially. If an error is detected in any register, the test is halted at that point and an error is returned. At the same time, a bitmask value representing the register where the error occurred is also returned.

- This API is intended to be called in privileged state (not user state).
- This API must be called with interrupts disabled. Registers that are non-volatile across function calls in C language are saved before and after the test.
- This API does not use interrupts.

Source files: SelfTest_fpu_register.s
Header files: SelfTest_fpu_register.h

Function name	
bool SelfTest_FPU_Register(uint32_t *result_word)	
Explanation	
Save the registers. Execute the test for FPU Extended RegistersS0～S31. Write 0x5555_5555 to S0. Compare S0 with 0x5555_5555, and if there is a mismatch, write 1 to bit0 of R0 and jump to error handling. Write 0xAAAA_AAAA to S0. Compare S0 with 0xAAAA_AAAA, and if there is a mismatch, write 1 to bit0 of R0 and jump to error handling. Write 0x5555_5555 to S1. Compare S1 with 0x5555_5555, and if there is a mismatch, write 1 to bit1 of R0 and jump to error handling. Write 0xAAAA_AAAA to S1. Compare S1 with 0xAAAA_AAAA, and if there is a mismatch, write 1 to bit1 of R0 and jump to error handling. Continue similarly for S2, S3, S4... S31. If there is a mismatch, write 1 to the corresponding bit in Output Parameters and jump to error handling. Restore the saved registers. Set R0 to Output Parameters. If the argument Output Parameters is not set, the Return Value (R0) will be 0x01. When the test is complete, the Return Value (R0) will be 0x00.	
Input Parameters	
None	
Output Parameters	
uint32_t *result_word	Memory buffer to store the results bit0: S0 Test Failed – bit31: S31 Test Failed
Return Value	
Bool	Test Result (true: Success, false: Failed)

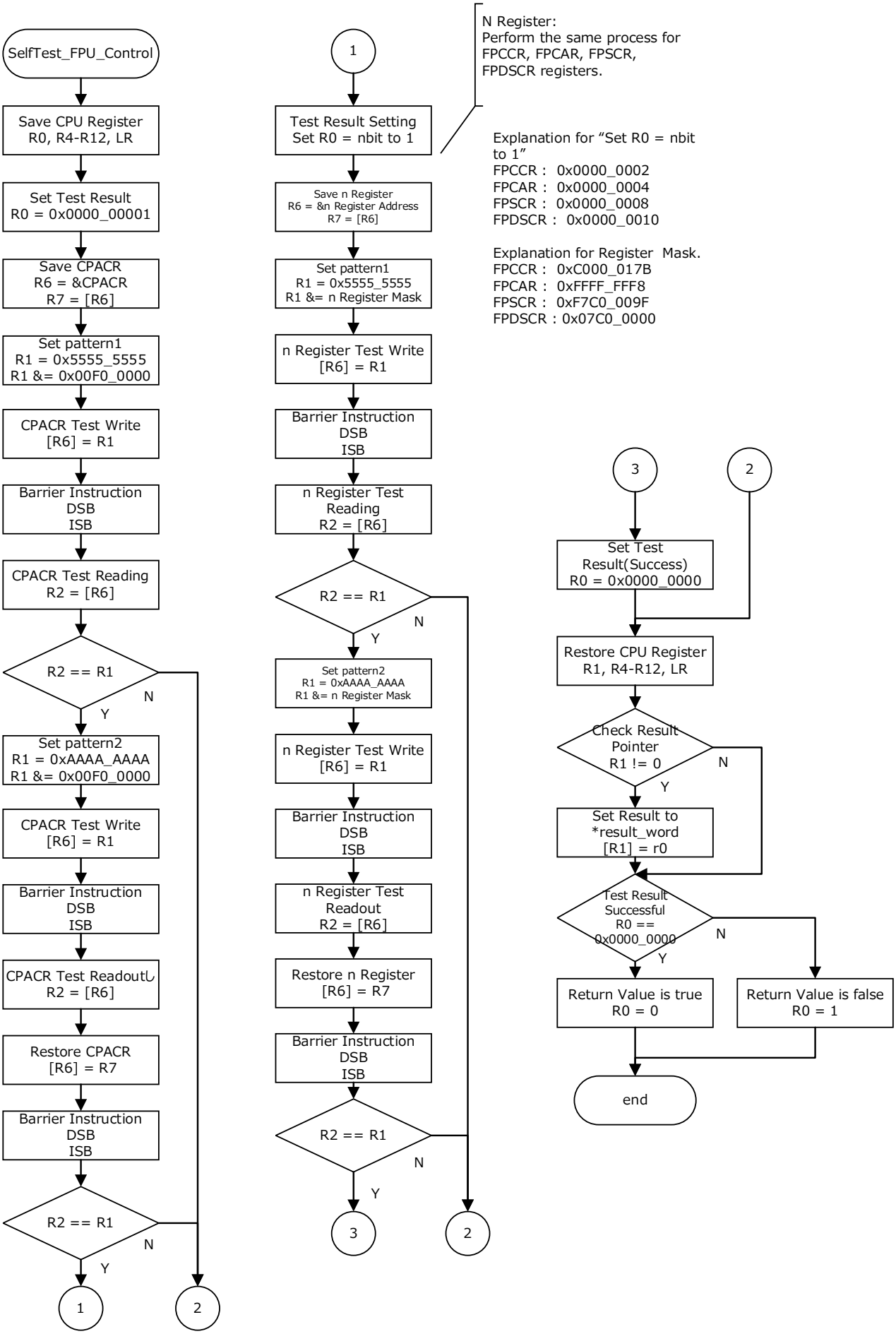
- Test Flowchart



Source files: SelfTest_fpu_control.s
Header files: SelfTest_fpu_control.h

Function name	
bool SelfTest_FPU_Control (uint32_t *result_word)	
Explanation	
Save the registers. Execute the test for FPU Control registers (CPACR, FPCCR, FPCAR, FPSCR, FPDSCR). Save the CPACR. Write 0x5555_5555 to the CPACR. Compare the CPACR with 0x5555_5555, and if there is a mismatch, write 1 to bit0 of R0. Restore the CPACR and jump to error handling. Write 0xAAAA_AAAA to the CPACR. Compare the CPACR with 0xAAAA_AAAA, and if there is a mismatch, write 1 to bit0 of R0. Restore the CPACR and jump to error handling. If the test is successful, restore the CPACR and execute the FPCCR test. Subsequently, execute the tests for FPCCR, FPCAR, FPSCR, and FPDSCR in order. If there is a mismatch, write 1 to the corresponding bit in Output Parameters and jump to error handling. Restore the saved registers. Set R0 to Output Parameters. If the argument Output Parameters is not set, the Return Value (R0) will be 0x01. When the test is complete, the Return Value (R0) will be 0x00.	
Input Parameters	
None	
Output Parameters	
uint32_t *result_word	Memory buffer to store the results bit0: CPACR Test Failed bit1: FPCCR Test Failed bit2: FPCAR Test Failed bit3: FPSCR Test Failed bit4: FPDSCR Test Failed
Return Value	
Bool	Test Result (true: Success, false: Failed)

- Test Flowchart



3.2.3. CPU Program Counter Test

Perform the "Stuck at" error check for the CPU program counter.
Instead of assigning values to the program counter within this function, multiple jump instructions are executed. If the test fails, the program will not recover. Conversely, if execution completes, the test is considered successful.

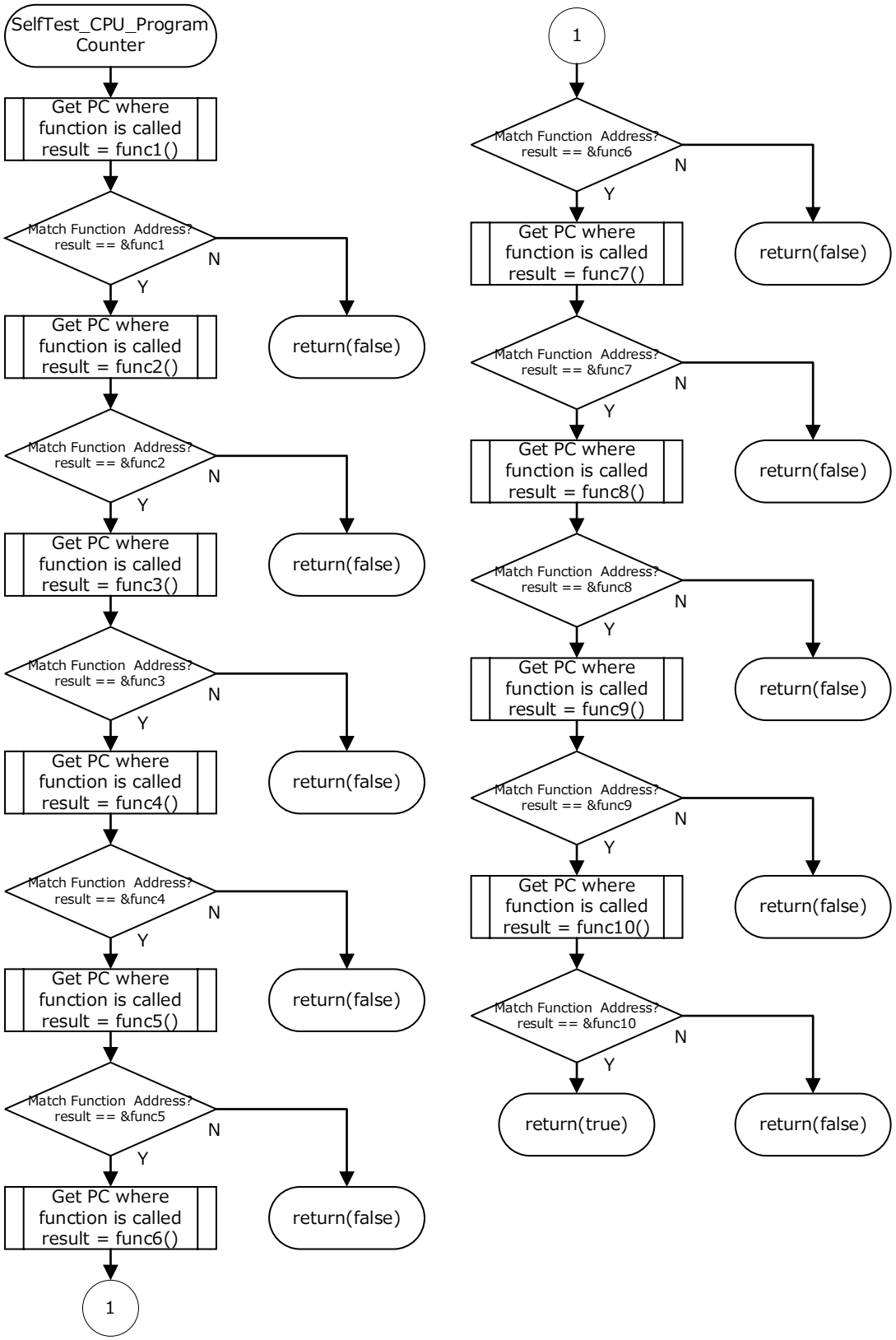
- This library function must be called with interrupts disabled. Registers that are non-volatile across function calls in C language are saved before and after the test.
- This library function does not use interrupts.

Source files: SelfTest_cpu.pc.c
Header files: SelfTest_cpu_pc.h

Function name	
bool SelfTest_CPU_ProgramCounter(void)	
Explanation	
Call the function (func1). The function (func1) retrieves the program counter (PC) and returns it as the Return Value. Compare the address of the function (func1) with the Return Value (PC), and if there is a mismatch, return failure (false) and terminate the test. If successful, call the function (func2). The function (func2) retrieves the program counter (PC) and returns it as the Return Value. Compare the address of the function (func2) with the Return Value, and if there is a mismatch, return failure (false) and terminate the test. If successful, call the function (func3). Continue similarly for functions (func3, func4... func10). If the function (func10) is successful, return success (true) and terminate the test.	
Input Parameters	
None	
Output Parameters	
None	
Return Value	
Bool	Test Result When executed successfully, true: success is returned. In case of failure, it generally does not return to the caller, but if false is returned, it may indicate an internal error such as improper compilation(Note1).

Note1: This can happen when the called subroutine is inlining. In this test implementation, function inlining is disabled.

- Test Flowchart



3.2.4. Interrupt Test

Perform the interrupt test using the timer as the interrupt source. An error is indicated if no interrupts occur during the test, or if the number of interrupts is significantly less or more than expected.

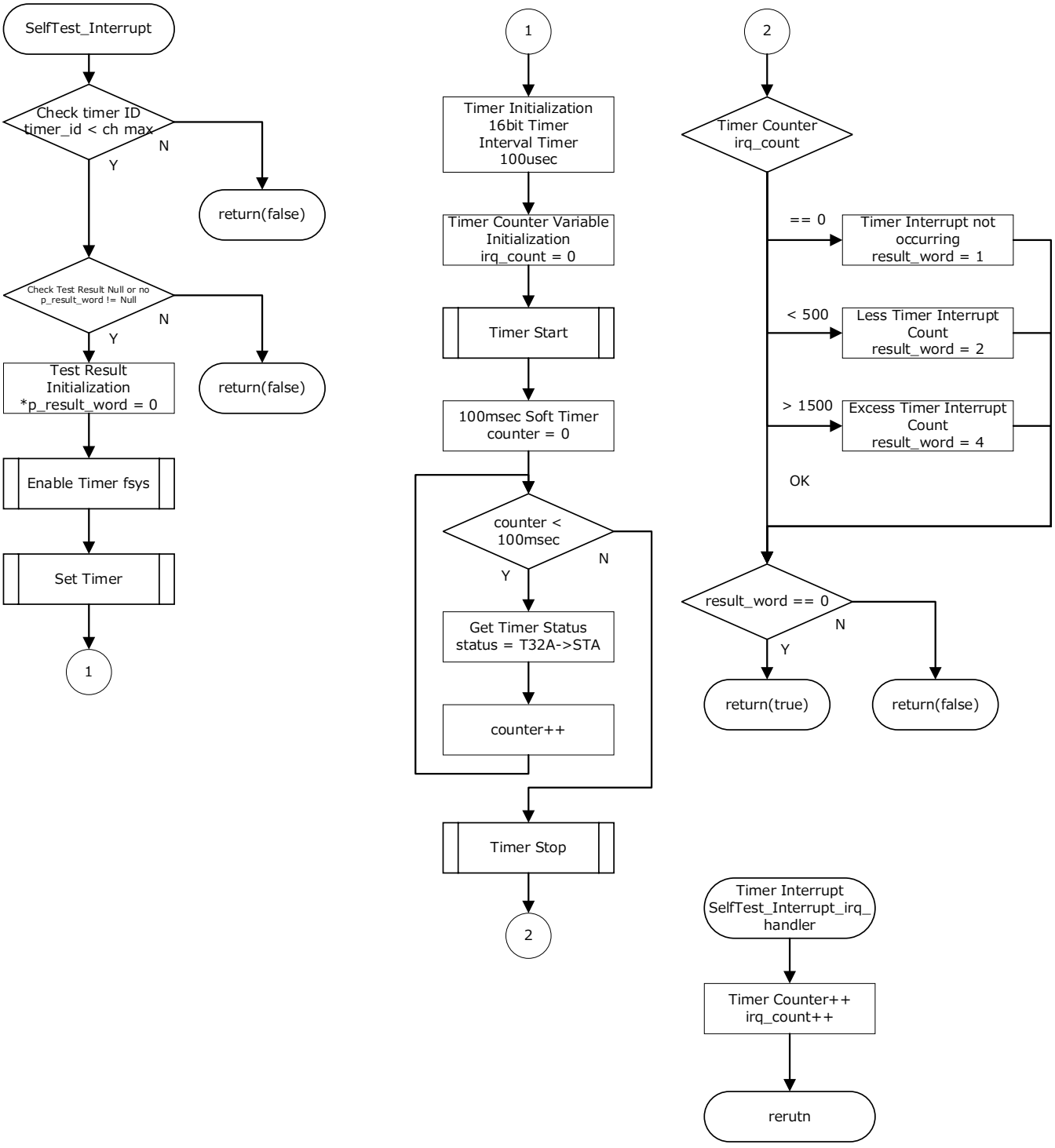
- The test duration is measured using a busy loop. Using a pre-measured loop counter value, the number of interrupts occurring at 100-microsecond intervals over approximately 100 milliseconds is counted. Typically, around 1000 interrupts should occur, but if the detected number of interrupts is between 500 and 1500, it is considered OK. If it falls outside this range or no interrupts occur, it is considered an error.

  - This test is intended to be executed before the initial setup of interrupt handling during system startup.
  - The busy loop duration depends on the microcontroller's operating frequency and can be adjusted by the user.
  - The OK detection condition for the number of occurrences can also be adjusted by the user.

Source files: SelfTest_interrupt.c
Header files:SelfTest_interrupt.h

Function name	
bool SelfTest_Interrupt(int timer_id, uint32_t *result_word)	
Explanation	
Initialize the T32 timer channel number specified by Input Parameters (timer_id). 16-bit timer mode, interval timer, 100 microsecond interrupt. Initialize the counter variable to 0 within the timer interrupt. Start the T32 timer count. Wait for 100 milliseconds using a software timer. Stop the T32 timer. If the counter variable is 0, set bit0 of Output Parameters to 1, return failure (false), and terminate the test. If the counter variable is less than 500, set bit1 of Output Parameters to 1, return failure (false), and terminate the test. If the counter variable is more than 1500, set bit2 of Output Parameters to 1, return failure (false), and terminate the test. If the counter variable is between 500 and 1500, set bits0, 1, and 2 of Output Parameters to 0, return success (true), and terminate the test.	
Input Parameters	
int timer_id	The T32 timer channel number(0~5) to be used.
Output Parameters	
uint32_t *result_word	Memory buffer to store the results bit0: No timer interrupt occurred. bit1: The number of timer interrupts is too few. bit2: The number of timer interrupts is too many.
Return Value	
Bool	Test Result(true: Success, false: Failed)

- Test Flowchart



3.2.5. Clock Frequency Test

Using the frequency detection circuit (OFD) and reference clock input (IHOSC2), check for clock frequency abnormalities in the system clock (fc) or external high-speed oscillator (fEHOSC) over a fixed period (100 ms).

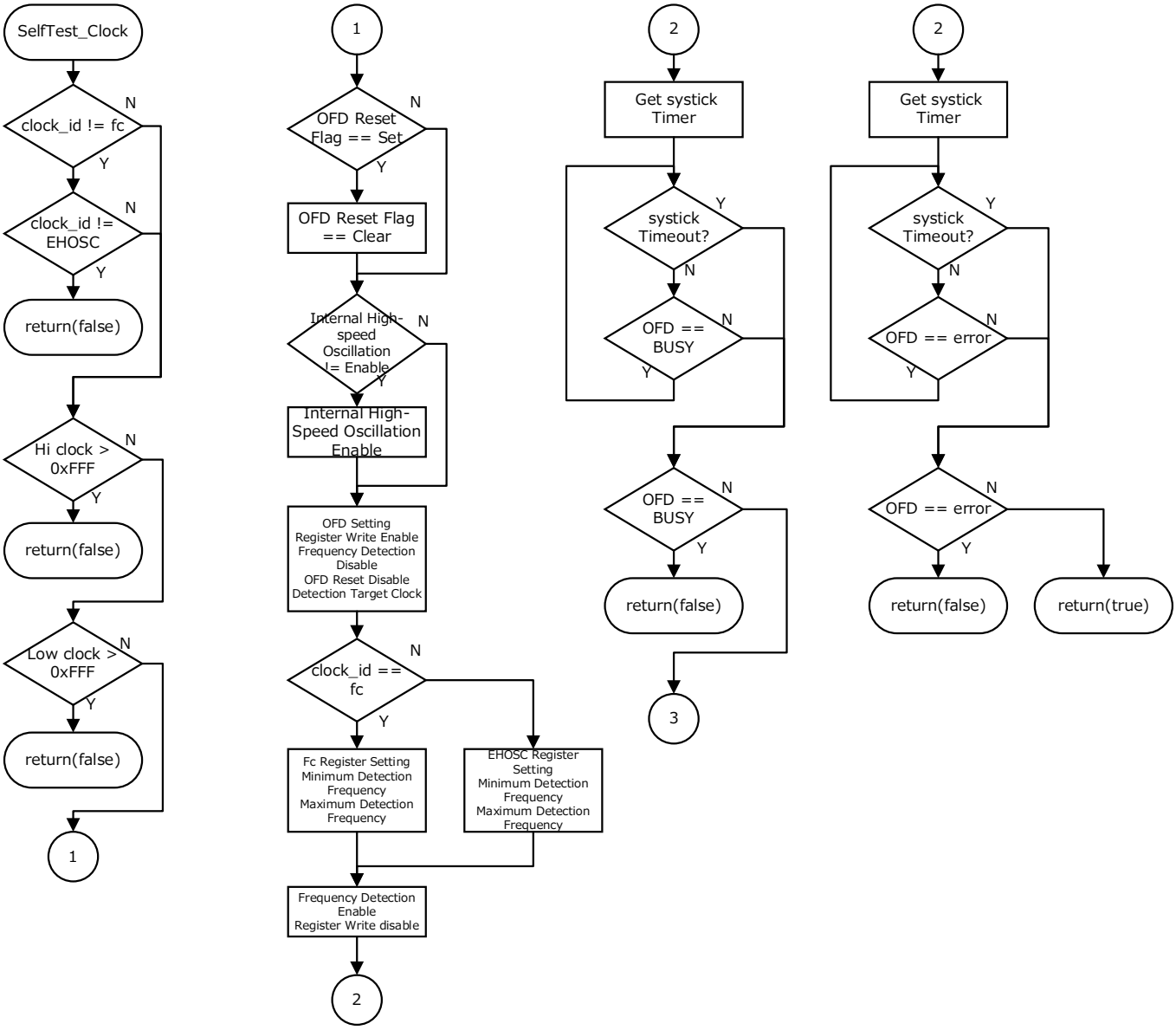
- The test clock can be selected from either fc or fEHOSC.
- If the reference clock (IHOSC2) is disabled, it will be automatically enabled during the test and returned to its original setting (enabled or disabled) after the test.

  - Example implementation:
    The frequency of the reference clock is 10 MHz. An example of calculating the values for high_clock and low_clock using this value is provided in the "OFD Reference Manual".

Source files: SelfTest_clock.c
Header files: SelfTest_clock.h

Function name	
bool SelfTest_clock(int clock_id, int high_clock, int low_clock)	
Explanation	
Check the settings of Input Parameters. If Input Parameters (clock_id) is not 0 (fc) or 1 (fEHOSC), return failure (false) and terminate the test. If Input Parameters (high_clock) is greater than 0x1000, return failure (false) and terminate the test. If Input Parameters (low_clock) is greater than 0x1000, return failure (false) and terminate the test. If Input Parameters are valid, Set the type of clock input to be tested, set the maximum detection frequency, and set the minimum detection frequency. Enable frequency detection operation control. Wait until the OFD operation state is active. Check the abnormal detection flag for 100 milliseconds. If the abnormal detection flag indicates an abnormality, return failure (false) and terminate the test. If the abnormal detection flag does not indicate an abnormality, return success (true) and terminate the test.	
Input Parameters	
int clock_id	Type of clock input to be tested: 0: fc 1: fEHOSC Any other value will result in an error.
int high_clock	High-frequency side setting value (12-bit). If the value is greater than 0x1000, it will result in an error.
int low_clock	Low-frequency side setting value (12-bit). If the value is greater than 0x1000, it will result in an error.
Output Parameters	
None	
Return Value	
bool	Test Result (true: Success, false: Failed)

- Test Flowchart



3.2.6. RAM Test

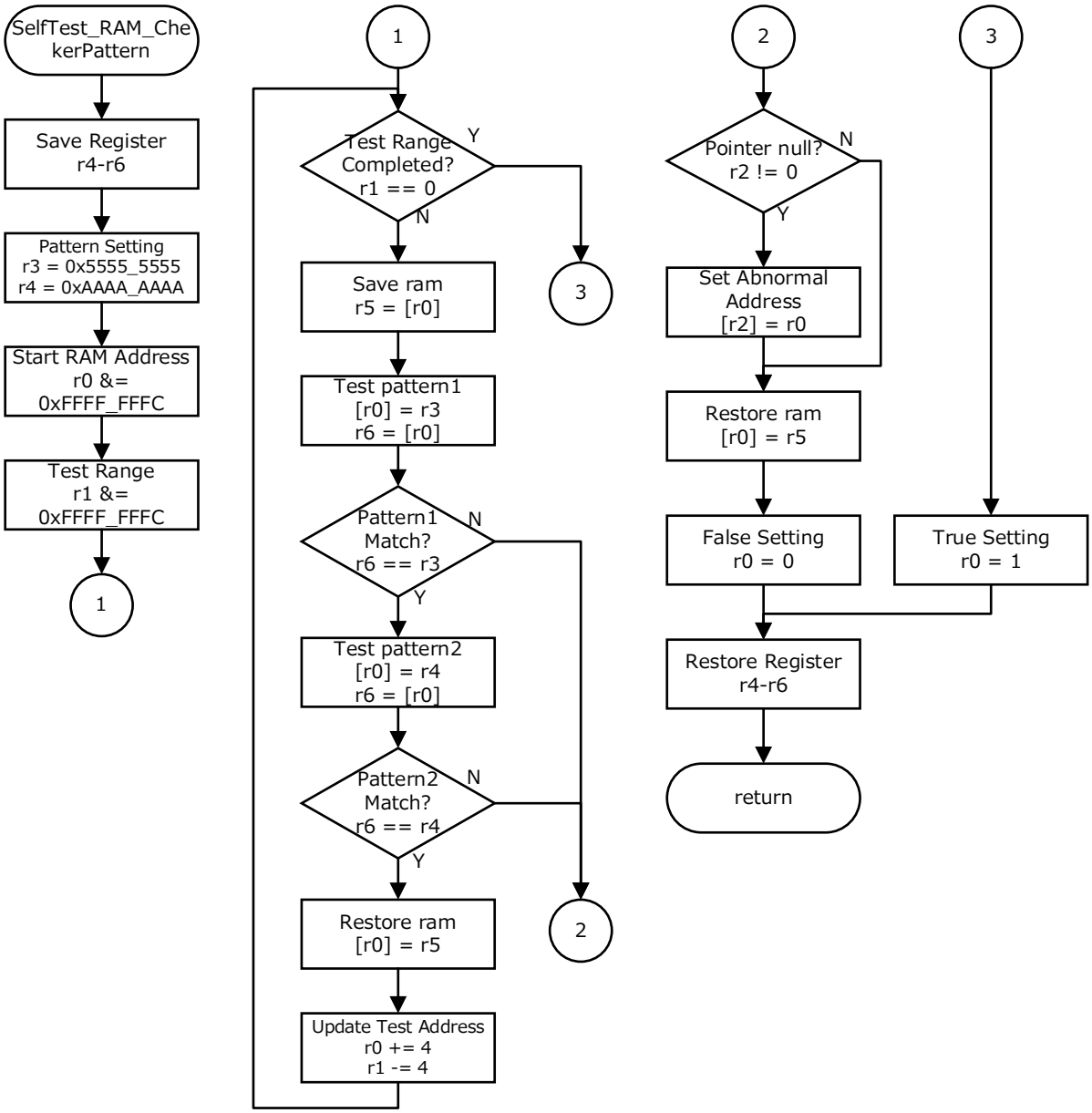
Verify that the checker patterns 0x55555555 and 0xaaaaaaaa can be correctly written and read at the same memory address using the March C algorithm. Execute this for the specified memory range.

- If an error is detected, write the address to the result memory buffer.
  - The value of each memory address is restored to its pre-test value after the test, but if the test fails, it cannot be guaranteed that the original value is restored.
  - If the test range includes RAM areas that are modified during interrupt processing (such as stack areas or variable areas written during interrupt processing), unexpected changes may occur. In such cases, it is recommended to divide the test area and execute the test for the problematic RAM area with interrupts disabled.
  - This API does not use interrupts.
  - If the device has RAMP functionality, verify using the RAMP functionality.

Source files: SelfTest_ram_checker.s
Header files: SelfTest_ram_checker.h

Function name	
bool SelfTest_RAM_Checker(uint32_t start, uint32_t length, uint32_t *result_word)	
Explanation	
Save the registers. Set the lower 2 bits of Input Parameters (start) to 0 (round to a multiple of 4). Set the lower 2 bits of Input Parameters (length) to 0 (round to a multiple of 4). Write 0x5555_5555 to the RAM address specified by start (*start = 0x55555555). Read the value from the RAM address specified by start. Compare the read value with 0x5555_5555, and if there is a mismatch, write the RAM address specified by stat to Output Parameters *result_word, return failure (false), and terminate the test. If they match, Write 0xAAAA_AAAA to the RAM address specified by start (*start = 0xAAAAAAAA). Read the value from the RAM address specified by start. Compare the read value with 0xAAAA_AAAA, and if there is a mismatch, write the RAM address specified by stat to Output Parameters *result_word, return failure (false), and terminate the test. If they match, increment the RAM address specified by start (start += 0x0000_00004). Decrement Input Parameters (length) (length -= 0x0000_00004). Repeat the comparison of 0x5555_5555 and 0xAAAA_AAAA until length becomes 0. If all comparisons match until length becomes 0, return success (true) and terminate the test.	
Input Parameters	
uint32_t start	The RAM address to start the test. It is rounded to a multiple of 4.
uint32_t length	The test range (in bytes). It is rounded to a multiple of 4.
Output Parameters	
uint32_t *result_word	Memory buffer to store the results Write the address where an abnormality is detected (do nothing if normal).
Return Value	
bool	Test Result (true: Success, false: Failed)

- Test Flowchart

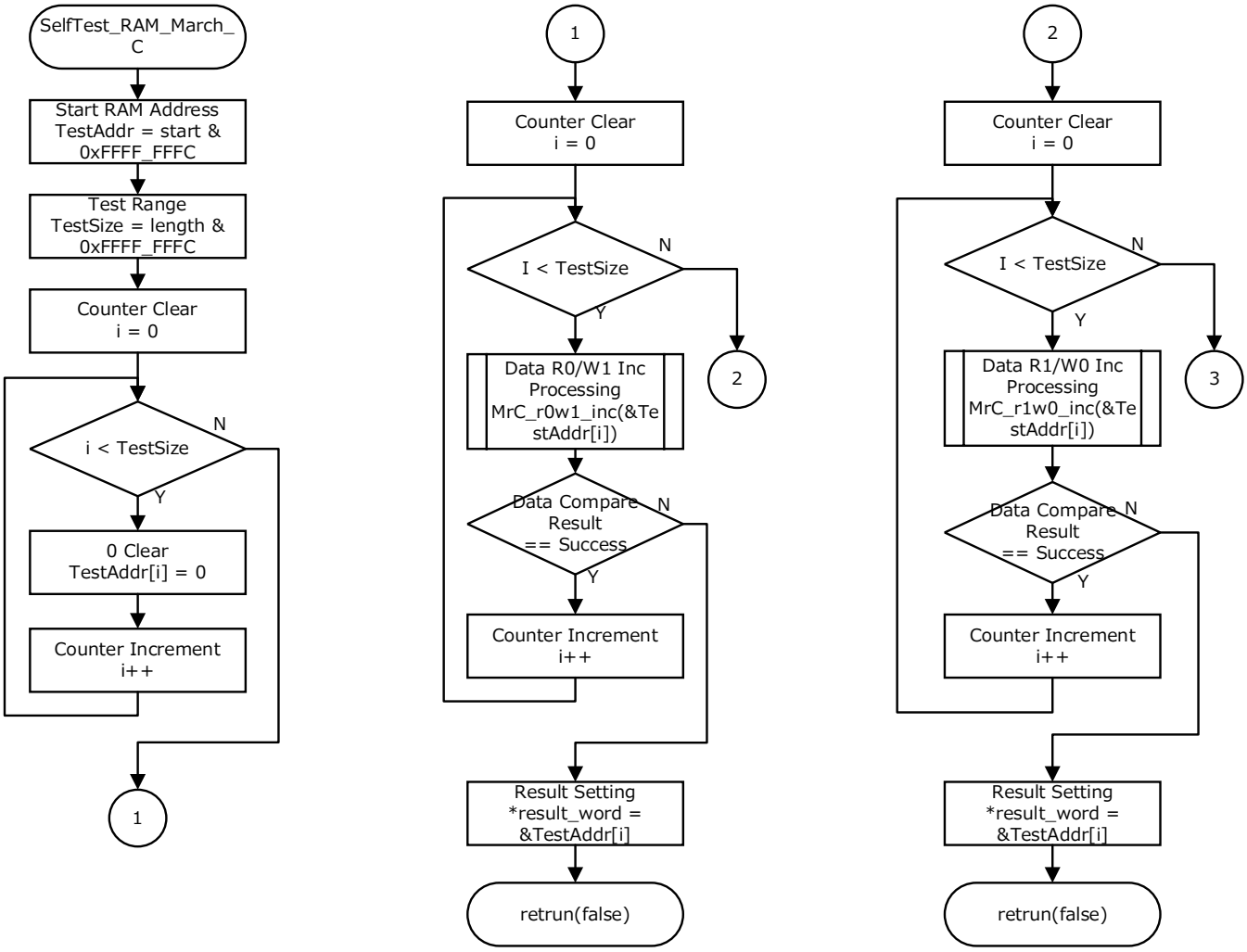


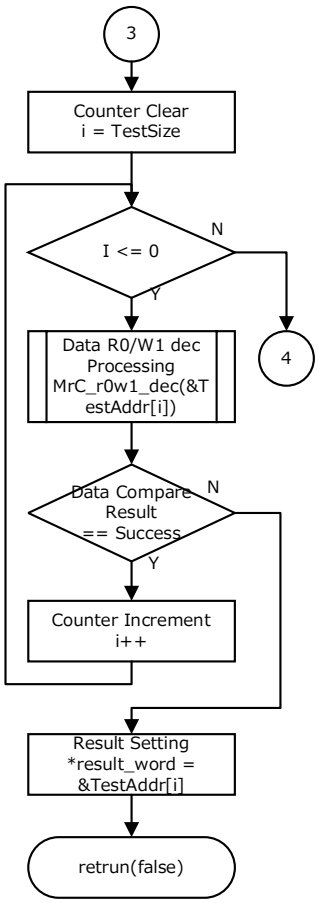
Source files: SelfTest_ram_march_c.s
Header files: SelfTest_ram_march_c.h

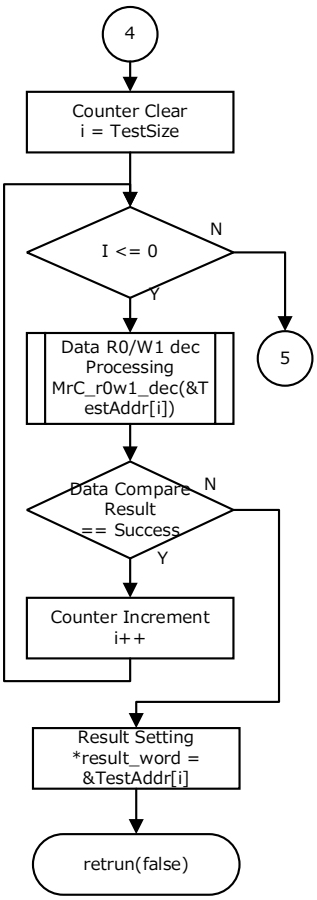
Function name
bool SelfTest_RAM_March_C(uint32_t start, uint32_t length, uint32_t *result_word)
Explanation
Save the registers. Set the lower 2 bits of Input Parameters (start) to 0 (round to a multiple of 4). Set the lower 2 bits of Input Parameters (length) to 0 (round to a multiple of 4). Write 0x0000_0000 to the RAM address specified by start for the size specified by length. Read the value from the RAM address specified by start and compare it with 0x0000_0000. If there is a mismatch, write the RAM address to Output Parameters *result_word, return failure (false), and terminate the test. If they match, write 0x0000_0001 to the RAM address. Read the value from the RAM address and compare it with 0x0000_0001. If there is a mismatch, write the RAM address to Output Parameters *result_word, return failure (false), and terminate the test. If they match, write 0x0000_0003 to the RAM address. Read the value from the RAM address and compare it with 0x0000_0003. If there is a mismatch, write the RAM address to Output Parameters *result_word, return failure (false), and terminate the test. If they match, write 0x0000_0007 to the RAM address. Read the value from the RAM address and compare it with 0x0000_0007. If there is a mismatch, write the RAM address to Output Parameters *result_word, return failure (false), and terminate the test. Continue similarly for 32 bits, one bit at a time. If all 32 bits match, increment the RAM address. Repeat for the specified size. If all comparisons match, read the RAM address and compare it with 0x1111_1111. If there is a mismatch, write the RAM address to Output Parameters *result_word, return failure (false), and terminate the test. If they match, write 0xFFFF_FFFE to the RAM address. Read the value from the RAM address and compare it with 0xFFFF_FFFE. If there is a mismatch, write the RAM address to Output Parameters *result_word, return failure (false), and terminate the test. If they match, write 0xFFFF_FFFC to the RAM address. Read the value from the RAM address and compare it with 0xFFFF_FFFC. If there is a mismatch, write the RAM address to Output Parameters *result_word, return failure (false), and terminate the test. If they match, write 0xFFFF_FFF8 to the RAM address. Read the value from the RAM address and compare it with 0xFFFF_FFF8. If there is a mismatch, write the RAM address to Output Parameters *result_word, return failure (false), and terminate the test. Continue similarly for 32 bits, one bit at a time. If all 32 bits match, increment the RAM address. Repeat for the specified size. When all match, read the RAM address (specified by start RAM address + size specified by length) and compare it with 0x0000_0000. If they do not match, write the RAM address to Output Parameters *result_word, return failure (false), and end the test. If they match, write 0x0000_0001 to the RAM address. Read the RAM address and compare it with 0x0000_0001. If they do not match, write the RAM address to Output Parameters *result_word, return failure (false), and end the test. If they match, write 0x0000_0003 to the RAM address. Read the RAM address and compare it with 0x0000_0003. If they do not match, write the RAM address to Output Parameters *result_word, return failure (false), and end the test. If they match, write 0x0000_0007 to the RAM address. Read the RAM address and compare it with 0x0000_0007. If they do not match, write the RAM address to Output Parameters *result_word, return failure (false), and end the test. Repeat for 32 bits, one bit at a time. When all 32 bits match, decrement the RAM address. Repeat for the specified size. When all match, read the RAM address and compare it with 0x1111_1111. If they do not match, write the RAM address to Output Parameters *result_word, return failure (false), and end the test. If they match, write 0xFFFF_FFFE to the RAM address. Read the RAM address and compare it with 0xFFFF_FFFE. If they do not match, write the RAM address to Output Parameters *result_word, return failure (false), and end the test.

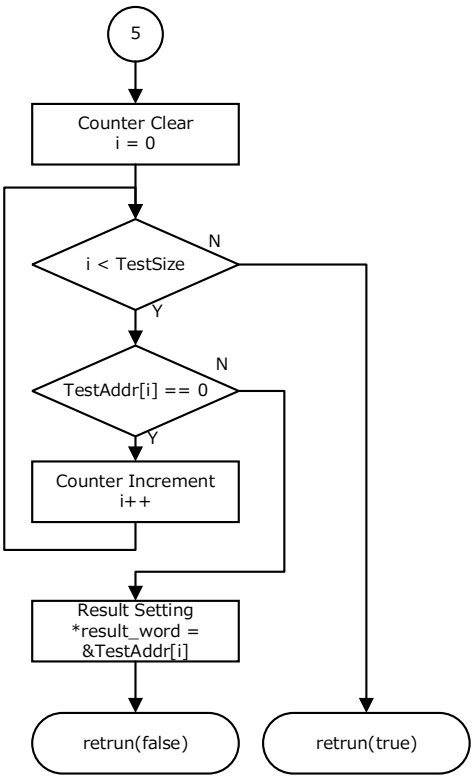
If they match, write 0xFFFF_FFFC to the RAM address. Read the RAM address and compare it with 0xFFFF_FFFC. If they do not match, write the RAM address to Output Parameters *result_word, return failure (false), and end the test. If they match, write 0xFFFF_FFF8 to the RAM address. Read the RAM address and compare it with 0xFFFF_FFF8. If they do not match, write the RAM address to Output Parameters *result_word, return failure (false), and end the test. Repeat for 32 bits, one bit at a time. When all 32 bits match, decrement the RAM address. Repeat for the specified size. When all match, return success (true) and end the test.	
Input Parameters	
uint32_t start	The RAM address to start the test Be rounded to a multiple of 4.
uint32_t length	The test range (in bytes) Be rounded to a multiple of 4.
Output Parameters	
uint32_t *result_word	Memory buffer to store the results Write the address where an anomaly is detected (do nothing if normal).
Return Value	
bool	Test Result (true: Success, false: Failed)

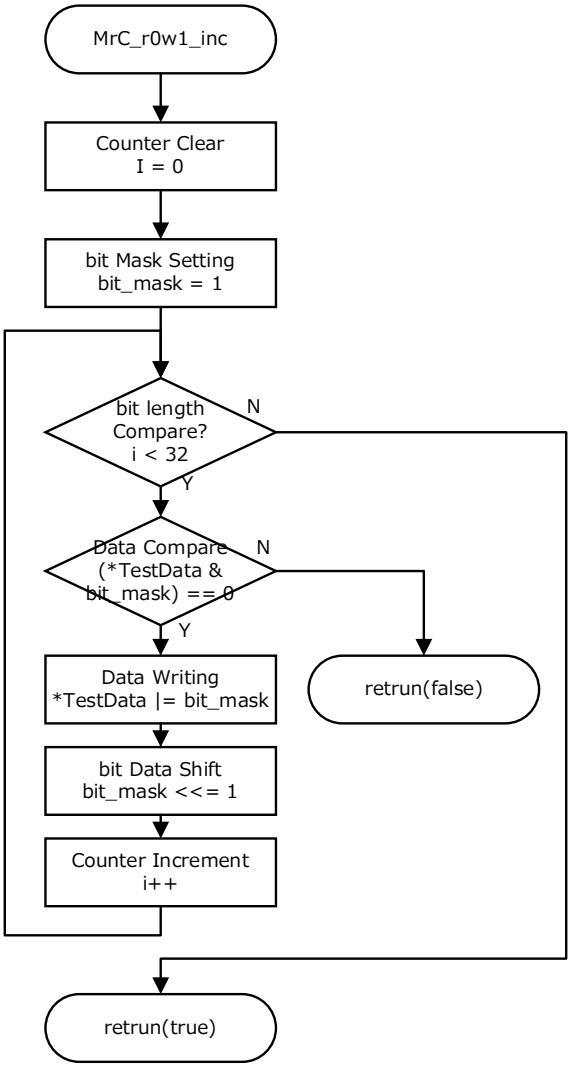
- Test Flowchart

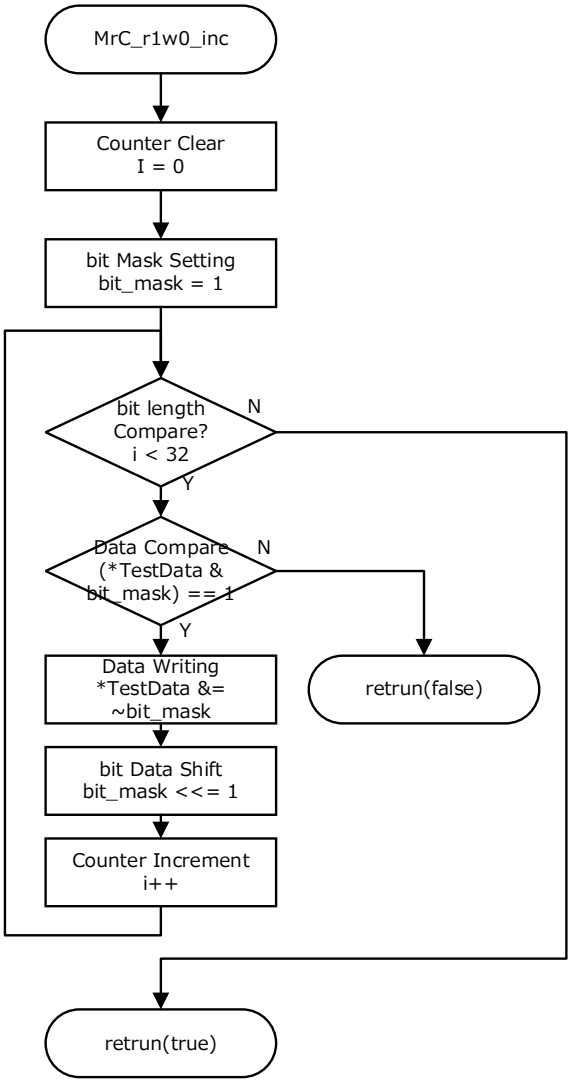


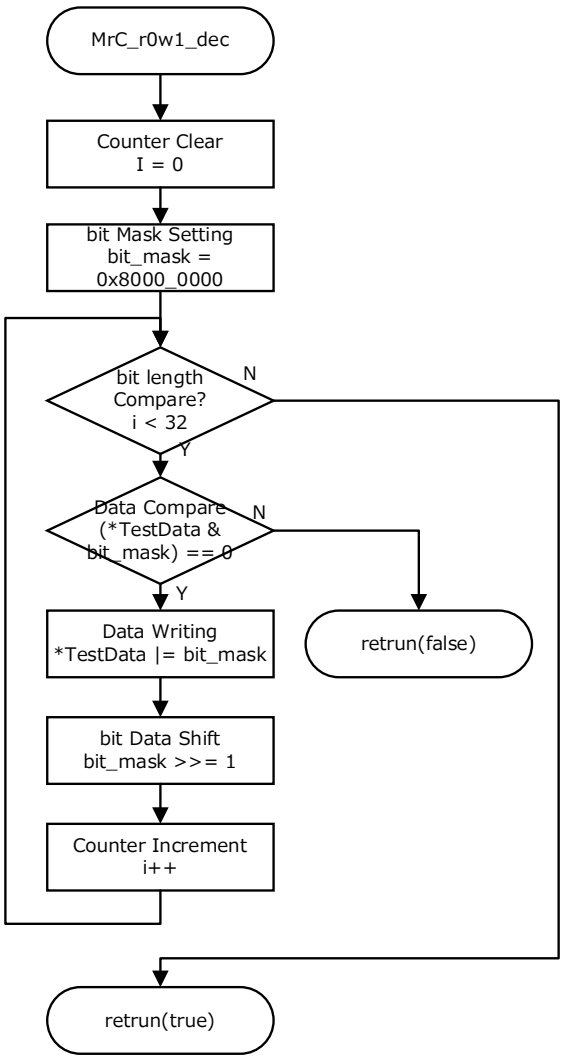

3
Counter Clear
i = TestSize
I <= 0
N
Y
4
Data R0/W1 dec
Processing
MrC_r0w1_dec(&T
estAddr[i])
Data Compare
Result
== Success
N
Y
Counter Increment
i++
Result Setting
*result_word =
&TestAddr[i]
retrun(false)

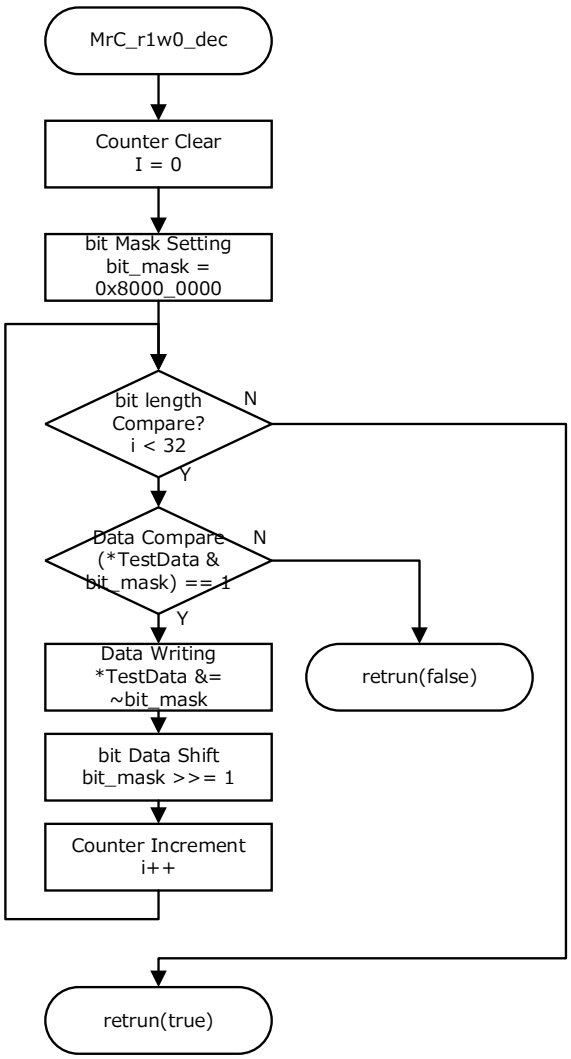

4
Counter Clear
i = TestSize
I <= 0
N
Y
5
Data R0/W1 dec
Processing
MrC_r0w1_dec(&T
estAddr[i])
Data Compare
Result
== Success
N
Y
Counter Increment
i++
Result Setting
*result_word =
&TestAddr[i]
retrun(false)


5
Counter Clear
i = 0
i < TestSize
N
Y
TestAddr[i] == 0
N
Y
Counter Increment
i++
Result Setting
*result_word =
&TestAddr[i]
retrun(false)
retrun(true)


MrC_r0w1_inc
Counter Clear
I = 0
bit Mask Setting
bit_mask = 1
bit length
Compare?
i < 32
N
Y
Data Compare
(*TestData &
bit_mask) == 0
N
Y
Data Writing
*TestData |= bit_mask
retrun(false)
bit Data Shift
bit_mask <<= 1
Counter Increment
i++
retrun(true)


MrC_r1w0_inc
Counter Clear
I = 0
bit Mask Setting
bit_mask = 1
bit length
Compare?
i < 32
N
Y
Data Compare
(*TestData &
bit_mask) == 1
N
Y
Data Writing
*TestData &=
~bit_mask
retrun(false)
bit Data Shift
bit_mask <<= 1
Counter Increment
i++
retrun(true)

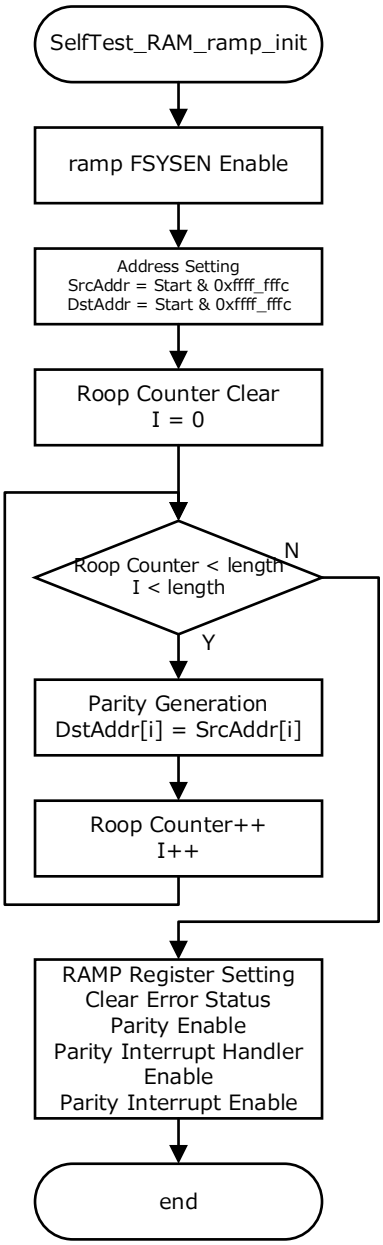

MrC_r0w1_dec
Counter Clear
I = 0
bit Mask Setting
bit_mask =
0x8000_0000
bit length
Compare?
i < 32
N
Y
Data Compare
(*TestData &
bit_mask) == 0
N
Y
Data Writing
*TestData |= bit_mask
retrun(false)
bit Data Shift
bit_mask >>= 1
Counter Increment
i++
retrun(true)


MrC_r1w0_dec
Counter Clear
I = 0
bit Mask Setting
bit_mask =
0x8000_0000
bit length
Compare?
i < 32
N
Y
Data Compare
(*TestData &
bit_mask) == 1
N
Y
Data Writing
*TestData &=
~bit_mask
retrun(false)
bit Data Shift
bit_mask >>= 1
Counter Increment
i++
retrun(true)

Source files: SelfTest_ram_ramp.c
Header files: SelfTest_ram_ramp.h

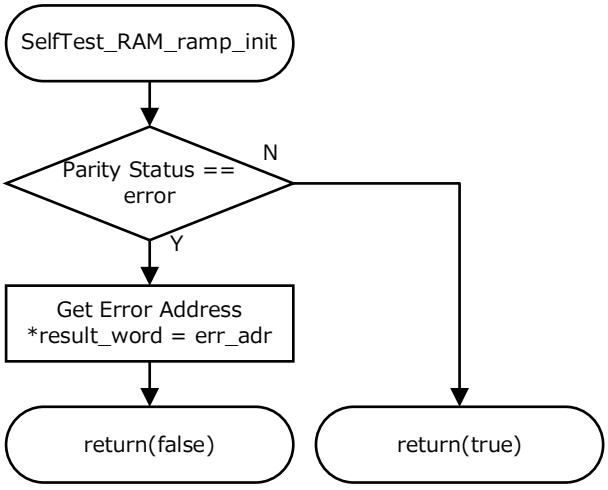
Function name	
void SelfTest_RAM_ramp_init(uint32_t start, uint32_t length, uint32_t)	
Explanation	
Enable the RAM parity function clock (fsys). Generate parity. Read the RAM address specified by Input Parameters (start) and write it to the same address. Increment the RAM address. Repeat for the size specified by Input Parameters (length). Clear the RAM parity function status. Enable the RAM parity function. Enable the RAM parity interrupt handler. When an anomaly is detected, jump to the RAM parity interrupt handler.	
Input Parameters	
uint32_t start	The RAM address to start the test Be rounded to a multiple of 4.
uint32_t length	The test range (in bytes) Be rounded to a multiple of 4.
Output Parameters	
---	---
Return Value	
---	---

- Test Flowchart



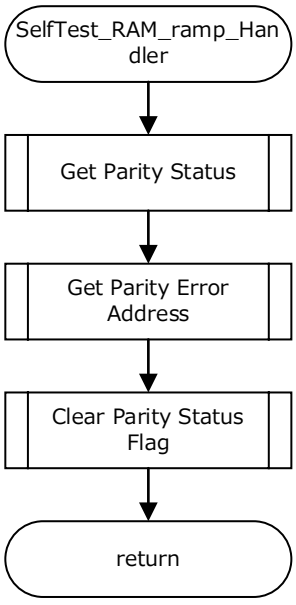
Function name	
bool SelfTest_RAM_ramp(*result_word)	
Explanation	
When an anomaly is detected in the RAM parity interrupt handler, write the address where the anomaly was detected to Output Parameters (*result_word), return failure (false), and end the test. If no anomaly is detected, return success (true) and end the test.	
Input Parameters	
---	---
---	---
Output Parameters	
uint32_t *result_word	Memory buffer to store the results Write the address where an anomaly is detected (do nothing if normal).
Return Value	
bool	Test Result (true: Success, false: Failed)

- Test Flowchart



Function name	
void SelfTest_RAM_ramp_Handler(void)	
Explanation	
When an anomaly is detected in the RAM parity interrupt handler, it is called by the RAM parity function. Check the RAM parity function status. If an error has occurred in the RAM parity function status, set the RAM parity error address to Output Parameters (*result_word) of SelfTest_RAM_ramp(). Clear the RAM parity function status and end the interrupt handler.	
Input Parameters	
---	---
---	---
Output Parameters	
---	---
Return Value	
---	---

- Test Flowchart



3.2.7. FLASH Test

Calculate the CRC32 value for the specified address range and ensure that the value read does not differ from the content written to the FLASH memory area.

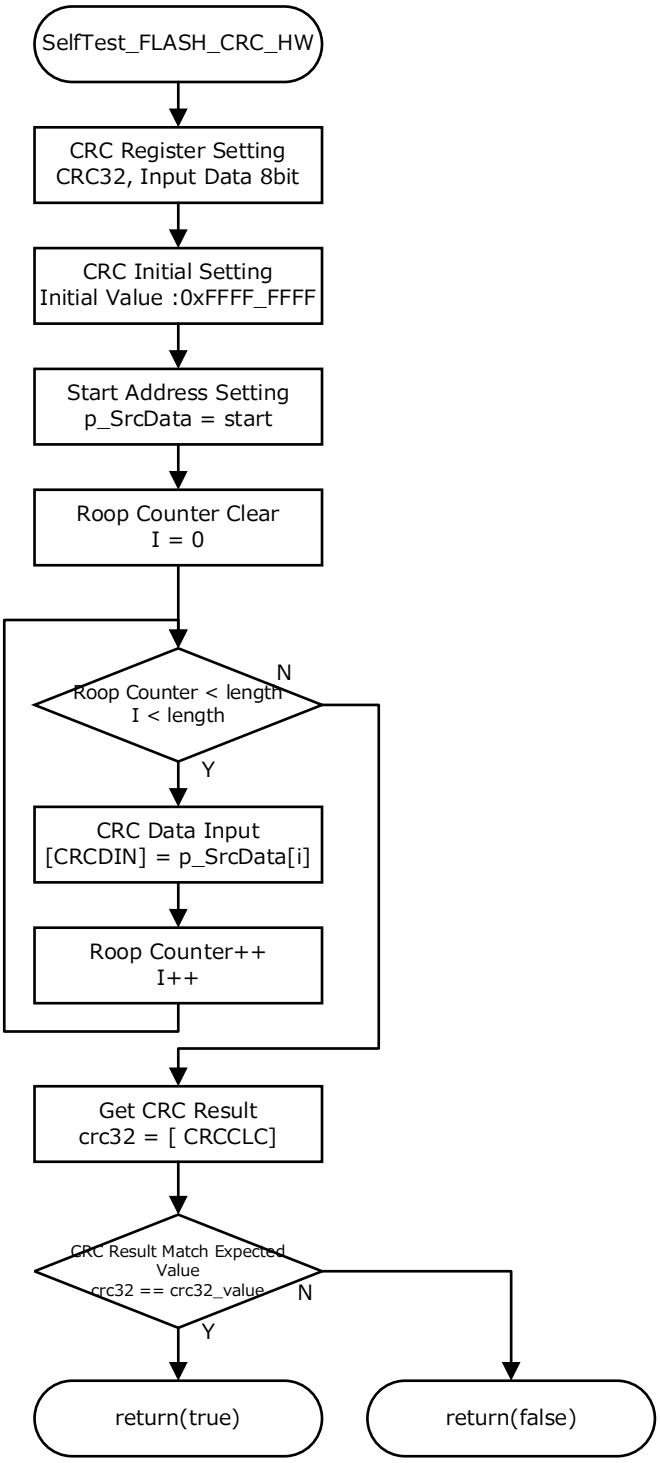
- This API does not check the values within the test range, allowing any range to be specified. The caller must correctly specify the appropriate range.

  - This API does not use interrupts.
  - To use this API, you need to pre-calculate the CRC32 value for a certain FLASH memory range.
  - If the device does not have a CRC function, calculate it using Software CRC

Source files: SelfTest_flash_crc_hw.c
Header files: SelfTest_flash_crc_hw.h

Function name	
bool SelfTest_FLASH_CRC_HW(uint32_t start, uint32_t length, uint32_t crc32_value)	
Explanation	
Initialize the CRC function. Set the CRC format (CRC32), input data width (8-bit), and CRC initial value result register (0xFFFF_FFFF). Read the FLASH address specified by Input Parameters (start) and write it to the CRC input data register. Increment the FLASH address and repeat for the size specified by Input Parameters (length). When the repetition for the specified size is complete, read the CRC initial value result register and compare it with Input Parameters (crc32_value). If they do not match, return failure (false) and end the test. If successful, return success (true) and end the test.	
Input Parameters	
uint32_t start	The FLASH address to start the test. Can be specified in 1-byte increments.
uint32_t length	The test range (in bytes) Can be specified in 1-byte increments.
uint32_t crc32_value	The pre-calculated CRC32 value for the test range.
Output Parameters	
None	
Return Value	
bool	Test Result (true: Success, false: Failed)

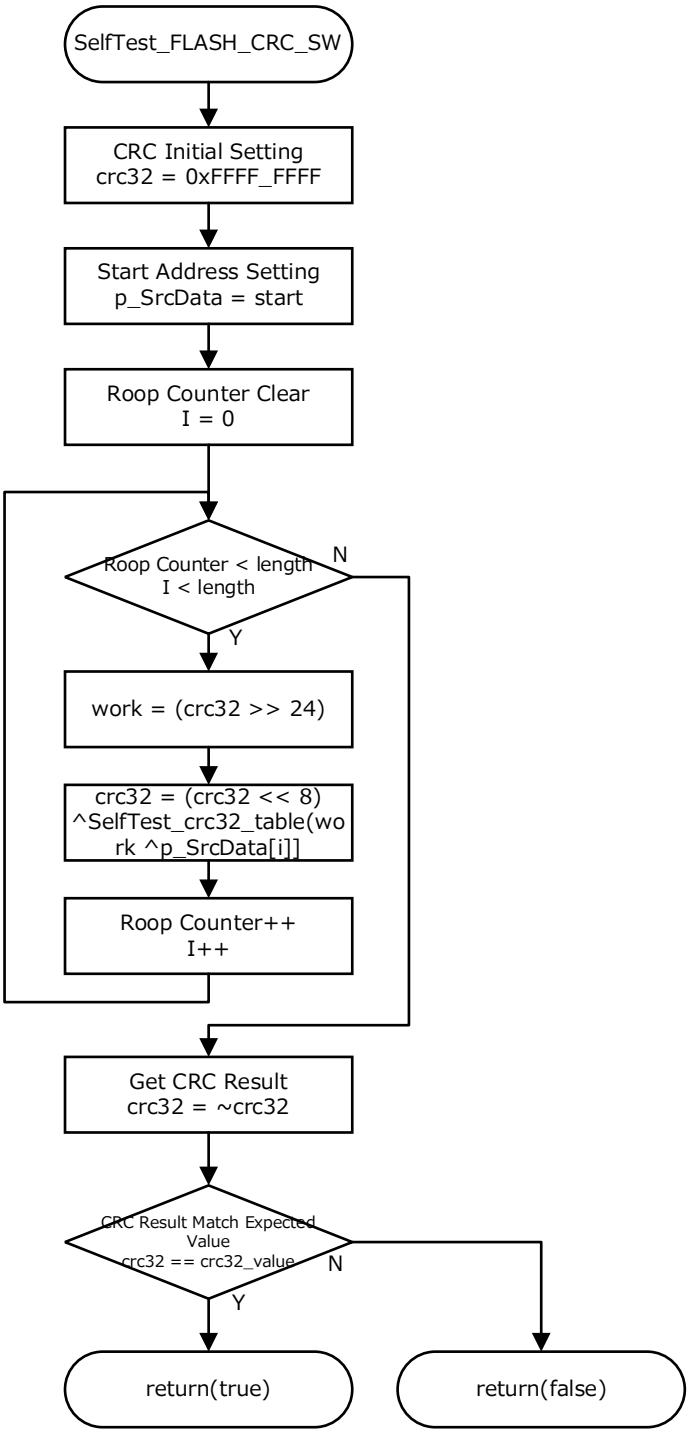
- Test Flowchart



Source files: SelfTest_flash_crc_sw.c
Header files: SelfTest_flash_crc_sw.h

Function name	
bool SelfTest_FLASH_CRC_SW(uint32_t start, uint32_t length, uint32_t crc32_value)	
Explanation	
Set the initial value of the variable crc to 0xFFFF_FFFF. Calculate for each 1byte. Work area = crc >> 24 FLASH data = FLASH address specified by Input Parameters (start). crc = CRC32 reference table [work area ^ FLASH data] ^ (crc << 8) Increment the FLASH address. Repeat for the size specified by Input Parameters (length). When the repetition for the specified size is complete, read the CRC initial value result register and compare it with Input Parameters (crc32_value). If they do not match, return failure (false) and end the test. If successful, return success (true) and end the test.	
Input Parameters	
uint32_t start	The FLASH address to start the test Can be specified in 1-byte increments.
uint32_t length	The test range (in bytes) Can be specified in 1-byte increments.
uint32_t crc32_value	The pre-calculated CRC32 value for the test range.
Output Parameters	
None	
Return Value	
bool	Test Result (true: Success, false: Failed)

- Test Flowchart



Use the reference table for CRC32.

```
uint32_t SelfTest_crc32_table[256] = {
    0x00000000UL, 0x04C11DB7UL, 0x09823B6EUL, 0x0D4326D9UL,
    0x130476DCUL, 0x17C56B6BUL, 0x1A864DB2UL, 0x1E475005UL,
    0x2608EDB8UL, 0x22C9F00FUL, 0x2F8AD6D6UL, 0x2B4BCB61UL,
    0x350C9B64UL, 0x31CD86D3UL, 0x3C8EA00AUL, 0x384FBDBDUL,
    0x4C11DB70UL, 0x48D0C6C7UL, 0x4593E01EUL, 0x4152FDA9UL,
    0x5F15ADACUL, 0x5BD4B01BUL, 0x569796C2UL, 0x52568B75UL,
    0x6A1936C8UL, 0x6ED82B7FUL, 0x639B0DA6UL, 0x675A1011UL,
    0x791D4014UL, 0x7DDC5DA3UL, 0x709F7B7AUL, 0x745E66CDUL,
    0x9823B6E0UL, 0x9CE2AB57UL, 0x91A18D8EUL, 0x95609039UL,
    0x8B27C03CUL, 0x8FE6DD8BUL, 0x82A5FB52UL, 0x8664E6E5UL,
    0xBE2B5B58UL, 0xBAEA46EFUL, 0xB7A96036UL, 0xB3687D81UL,
    0xAD2F2D84UL, 0xA9EE3033UL, 0xA4AD16EAUL, 0xA06C0B5DUL,
    0xD4326D90UL, 0xD0F37027UL, 0xDDB056FEUL, 0xD9714B49UL,
    0xC7361B4CUL, 0xC3F706FBUL, 0xCEB42022UL, 0xCA753D95UL,
    0xF23A8028UL, 0xF6FB9D9FUL, 0xFBB8BB46UL, 0xFF79A6F1UL,
    0xE13EF6F4UL, 0xE5FFEB43UL, 0xE8BCCD9AUL, 0xEC7DD02DUL,
    0x34867077UL, 0x30476DC0UL, 0x3D044B19UL, 0x39C556AEUL,
    0x278206ABUL, 0x23431B1CUL, 0x2E003DC5UL, 0x2AC12072UL,
    0x128E9DCFUL, 0x164F8078UL, 0x1B0CA6A1UL, 0x1FCDBB16UL,
    0x018AEB13UL, 0x054BF6A4UL, 0x0808D07DUL, 0x0CC9CDCAUL,
    0x7897AB07UL, 0x7C56B6B0UL, 0x71159069UL, 0x75D48DDEUL,
    0x6B93DDDBUL, 0x6F52C06CUL, 0x6211E6B5UL, 0x66D0FB02UL,
    0x5E9F46BFUL, 0x5A5E5B08UL, 0x571D7DD1UL, 0x53DC6066UL,
    0x4D9B3063UL, 0x495A2DD4UL, 0x44190B0DUL, 0x40D816BAUL,
    0xACA5C697UL, 0xA864DB20UL, 0xA527FDF9UL, 0xA1E6E04EUL,
    0xBFA1B04BUL, 0xBB60ADFCUL, 0xB6238B25UL, 0xB2E29692UL,
    0x8AAD2B2FUL, 0x8E6C3698UL, 0x832F1041UL, 0x87EE0DF6UL,
    0x99A95DF3UL, 0x9D684044UL, 0x902B669DUL, 0x94EA7B2AUL,
    0xE0B41DE7UL, 0xE4750050UL, 0xE9362689UL, 0xEDF73B3EUL,
    0xF3B06B3BUL, 0xF771768CUL, 0xFA325055UL, 0xFEF34DE2UL,
    0xC6BCF05FUL, 0xC27DEDE8UL, 0xCF3ECB31UL, 0xCBFFD686UL,
    0xD5B88683UL, 0xD1799B34UL, 0xDC3ABDEDUL, 0xD8FBA05AUL,
    0x690CE0EEUL, 0x6DCDFD59UL, 0x608EDB80UL, 0x644FC637UL,
    0x7A089632UL, 0x7EC98B85UL, 0x738AAD5CUL, 0x774BB0EBUL,
    0x4F040D56UL, 0x4BC510E1UL, 0x46863638UL, 0x42472B8FUL,
    0x5C007B8AUL, 0x58C1663DUL, 0x558240E4UL, 0x51435D53UL,
    0x251D3B9EUL, 0x21DC2629UL, 0x2C9F00F0UL, 0x285E1D47UL,
    0x36194D42UL, 0x32D850F5UL, 0x3F9B762CUL, 0x3B5A6B9BUL,
    0x0315D626UL, 0x07D4CB91UL, 0x0A97ED48UL, 0x0E56F0FFUL,
    0x1011A0FAUL, 0x14D0BD4DUL, 0x19939B94UL, 0x1D528623UL,
    0xF12F560EUL, 0xF5EE4BB9UL, 0xF8AD6D60UL, 0xFC6C70D7UL,
    0xE22B20D2UL, 0xE6EA3D65UL, 0xEBA91BBCUL, 0xEF68060BUL,
    0xD727BBB6UL, 0xD3E6A601UL, 0xDEA580D8UL, 0xDA649D6FUL,
    0xC423CD6AUL, 0xC0E2D0DDUL, 0xCDA1F604UL, 0xC960EBB3UL,
    0xBD3E8D7EUL, 0xB9FF90C9UL, 0xB4BCB610UL, 0xB07DABA7UL,
    0xAE3AFBA2UL, 0xAAFBE615UL, 0xA7B8C0CCUL, 0xA379DD7BUL,
    0x9B3660C6UL, 0x9FF77D71UL, 0x92B45BA8UL, 0x9675461FUL,
    0x8832161AUL, 0x8CF30BADUL, 0x81B02D74UL, 0x857130C3UL,
    0x5D8A9099UL, 0x594B8D2EUL, 0x5408ABF7UL, 0x50C9B640UL,
    0x4E8EE645UL, 0x4A4FFBF2UL, 0x470CDD2BUL, 0x43CDC09CUL,
    0x7B827D21UL, 0x7F436096UL, 0x7200464FUL, 0x76C15BF8UL,
    0x68860BFDUL, 0x6C47164AUL, 0x61043093UL, 0x65C52D24UL,
    0x119B4BE9UL, 0x155A565EUL, 0x18197087UL, 0x1CD86D30UL,
    0x029F3D35UL, 0x065E2082UL, 0x0B1D065BUL, 0x0FDC1BECUL,
    0x3793A651UL, 0x3352BBE6UL, 0x3E119D3FUL, 0x3AD08088UL,
    0x2497D08DUL, 0x2056CD3AUL, 0x2D15EBE3UL, 0x29D4F654UL,
    0xC5A92679UL, 0xC1683BCEUL, 0xCC2B1D17UL, 0xC8EA00A0UL,
```

```
    0xD6AD50A5UL, 0xD26C4D12UL, 0xDF2F6BCBUL, 0xDBEE767CUL,
    0xE3A1CBC1UL, 0xE760D676UL, 0xEA23F0AFUL, 0xEEE2ED18UL,
    0xF0A5BD1DUL, 0xF464A0AAUL, 0xF9278673UL, 0xFDE69BC4UL,
    0x89B8FD09UL, 0x8D79E0BEUL, 0x803AC667UL, 0x84FBDBD0UL,
    0x9ABC8BD5UL, 0x9E7D9662UL, 0x933EB0BBUL, 0x97FFAD0CUL,
    0xAFB010B1UL, 0xAB710D06UL, 0xA6322BDFUL, 0xA2F33668UL,
    0xBCB4666DUL, 0xB8757BDAUL, 0xB5365D03UL, 0xB1F740B4UL,
};
```

3.2.8. Digital Input/Output

Use any port to output digital data from GPIO, then read the value. Perform this for both 0 output and 1 output.

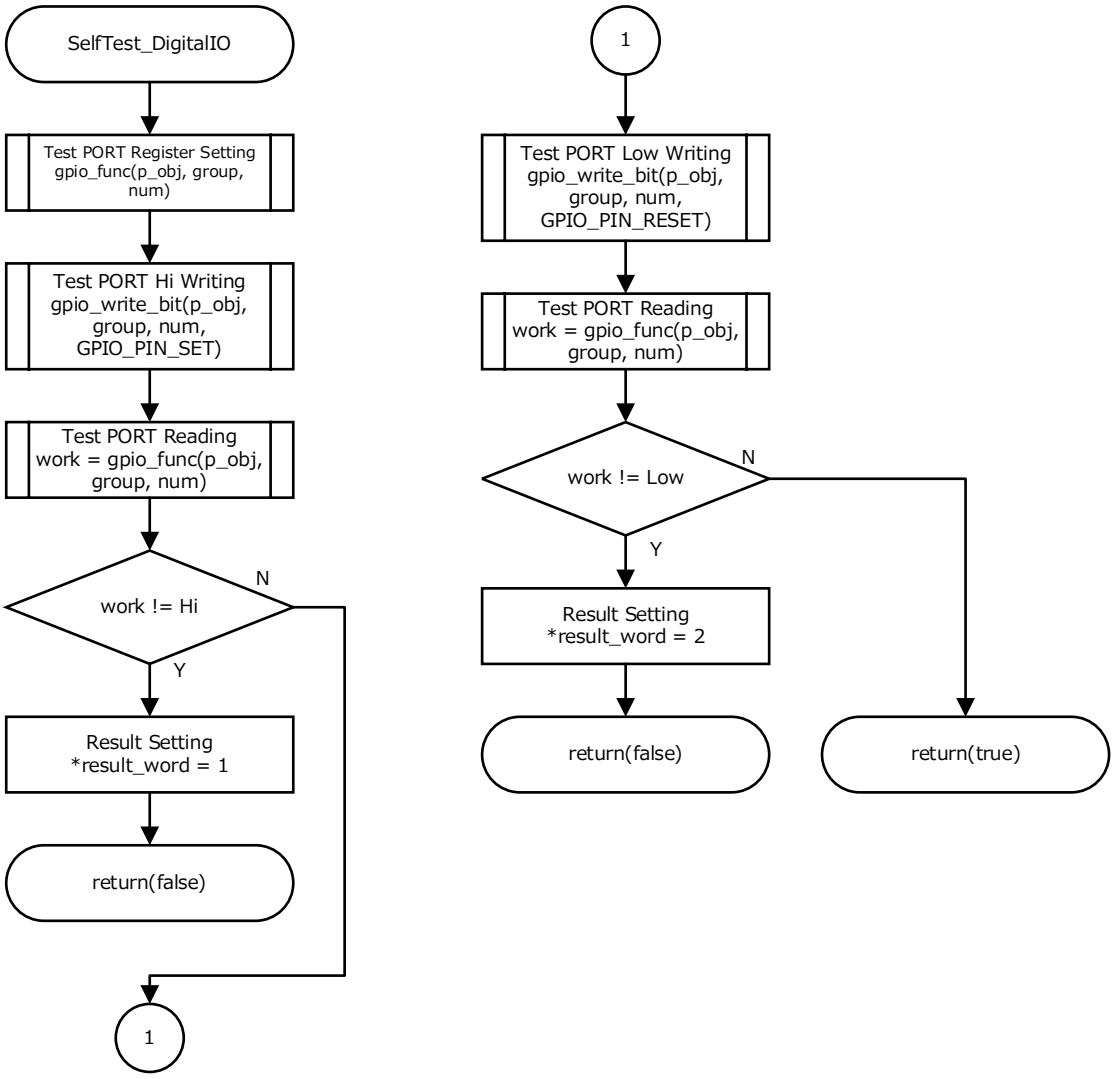
- Since the GPIO settings of the specified port will be changed after the test, either perform this test before starting the application or reconfigure the GPIO settings after the test.

  - This API does not use interrupts.
  - The user can specify any port.

Source files: SelfTest_digital_io.c
Header files: SelfTest_digital_io.h

Function name	
bool SelfTest_DigitalIO(DigitalIO_t port)	
Explanation	
Specify the port group, port number, and port function using constant definitions. Allow input and output for the specified port. Output '1' to the specified port. Read the input register of the specified port and compare it with '1'. If they do not match, set bit0 of Output Parameters (*result_word) to 1, return failure (false), and end the test. Output '0' to the specified port. Read the input register of the specified port and compare it with '0'. If they do not match, set bit1 of Output Parameters (*result_word) to 1, return failure (false), and end the test. If successful, return success (true) and end the test. Input Parameters and Output Parameters indicate the members of the argument DigitalIO_t structure.	
Input Parameters	
gpio_t *p_obj	The port instance to test
gpio_gr_t group	The port group to test
gpio_num_t num	The port number to test
Output Parameters	
uint32_t *result_word	When an error occurs, one of the following bits will be set: bit0: Error during '1' output bit1: Error during '0' output If an error is detected during any test, the test will be stopped and no further tests will be performed.
Return Value	
Bool	Result (true: Success, false: Failed)

- Test Flowchart



3.2.9. Analog Input Test

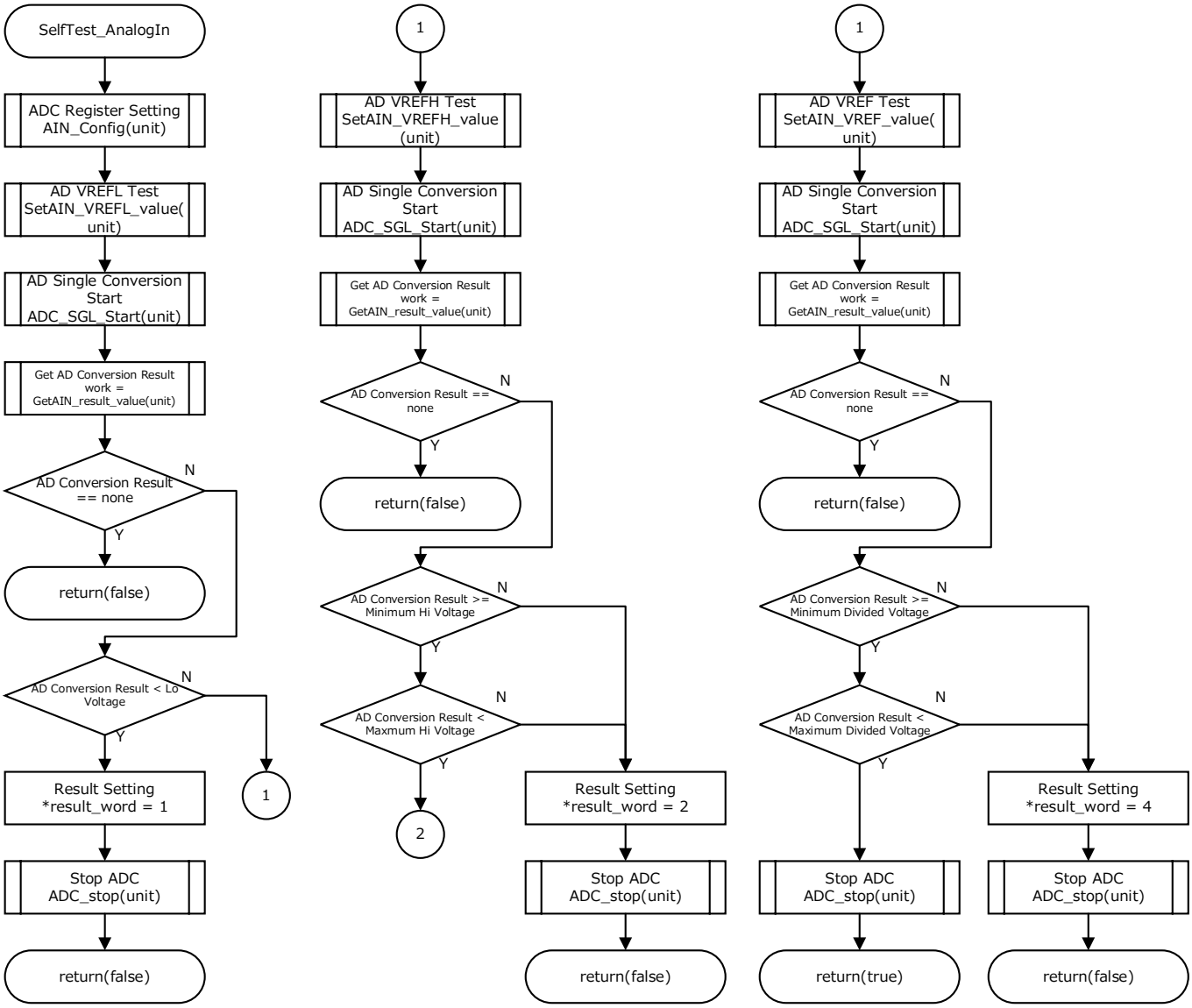
Use the ADC channel with test functionality to generate three types of input voltages for input verification (VREFL/VREFH/REGOUT1) and test the values obtained through AD conversion.

- This API does not use interrupts.
- The test functionality refers to hardware equipped with self-diagnosis support features.

Source files: SelfTest_analog_in.c
Header files: SelfTest_analog_in.h

Function name	
bool SelfTest_AnalogIn(uint32_t unit, uint32_t *result_word)	
Explanation	
Initialize the ADC function. Perform the low voltage test. Initial setup of the AD channel connected to VREFL. Start ADC single conversion. Obtain the conversion result of the AD channel connected to VREFL. Compare the conversion result with the low voltage. If it is below the low voltage, stop the ADC conversion, set bit0 of Output Parameters (*result_word) to 1, return failure (false), and end the test. If it is above the low voltage, perform the high voltage test. Initial setup of the AD channel connected to VREFH. Start ADC single conversion. Obtain the conversion result of the AD channel connected to VREFH. Compare the conversion result with the high voltage. If it is above the high voltage, stop the ADC conversion, set bit1 of Output Parameters (*result_word) to 1, return failure (false), and end the test. If it is below the high voltage, perform the divided voltage test. Initial setup of the AD channel connected to the reference power supply. Start ADC single conversion. Obtain the conversion result of the AD channel connected to the reference power supply. Compare the conversion result with the divided voltage. If it is more than 10% above or below the divided voltage, stop the ADC conversion, set bit2 of Output Parameters (*result_word) to 1, return failure (false), and end the test. If all tests are successful, stop the ADC conversion, return success (true), and end the test.	
Input Parameters	
uint32_t unit	The unit name of the ADC to be tested UNIT_A, UNIT_B, UNIT_C Other than the above will result in an error.
Output Parameters	
uint32_t *result_word	When an error occurs, one of the following bits will be set. bit0: Error during Lo voltage test bit1: Error during Hi voltage test bit2: Error during divided voltage test If an error is detected in any test, the test will be stopped and no further tests will be performed.
Return Value	
Bool	Result (true: Success, false: Failed)

- Test Flowchart



4. Revision History

Revision	Data	Description
1.0	2026/04/01	First release

RESTRICTIONS ON PRODUCT USE

Toshiba Corporation and its subsidiaries and affiliates are collectively referred to as “TOSHIBA”.
Hardware, software and systems described in this document are collectively referred to as “Product”.

- TOSHIBA reserves the right to make changes to the information in this document and related Product without notice.
- This document and any information herein may not be reproduced without prior written permission from TOSHIBA. Even with TOSHIBA's written permission, reproduction is permissible only if reproduction is without alteration/omission.
- Though TOSHIBA works continually to improve Product's quality and reliability, Product can malfunction or fail. Customers are responsible for complying with safety standards and for providing adequate designs and safeguards for their hardware, software and systems which minimize risk and avoid situations in which a malfunction or failure of Product could cause loss of human life, bodily injury or damage to property, including data loss or corruption. Before customers use the Product, create designs including the Product, or incorporate the Product into their own applications, customers must also refer to and comply with (a) the latest versions of all relevant TOSHIBA information, including without limitation, this document, the specifications, the data sheets and application notes for Product and the precautions and conditions set forth in the "TOSHIBA Semiconductor Reliability Handbook" and (b) the instructions for the application with which the Product will be used with or for. Customers are solely responsible for all aspects of their own product design or applications, including but not limited to (a) determining the appropriateness of the use of this Product in such design or applications; (b) evaluating and determining the applicability of any information contained in this document, or in charts, diagrams, programs, algorithms, sample application circuits, or any other referenced documents; and (c) validating all operating parameters for such designs and applications. **TOSHIBA ASSUMES NO LIABILITY FOR CUSTOMERS' PRODUCT DESIGN OR APPLICATIONS.**
- **PRODUCT IS NEITHER INTENDED NOR WARRANTED FOR USE IN EQUIPMENTS OR SYSTEMS THAT REQUIRE EXTRAORDINARILY HIGH LEVELS OF QUALITY AND/OR RELIABILITY, AND/OR A MALFUNCTION OR FAILURE OF WHICH MAY CAUSE LOSS OF HUMAN LIFE, BODILY INJURY, SERIOUS PROPERTY DAMAGE AND/OR SERIOUS PUBLIC IMPACT ("UNINTENDED USE").** Except for specific applications as expressly stated in this document, Unintended Use includes, without limitation, equipment used in nuclear facilities, equipment used in the aerospace industry, Class 3 medical devices, equipment used for automobiles, and military vehicles and munitions. **IF YOU USE PRODUCT FOR UNINTENDED USE, TOSHIBA ASSUMES NO LIABILITY FOR PRODUCT.** For details, please contact your TOSHIBA sales representative or contact us via our website.
- Do not disassemble, analyze, reverse-engineer, alter, modify, translate or copy Product, whether in whole or in part.
- Product shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable laws or regulations.
- The information contained herein is presented only as guidance for Product use. No responsibility is assumed by TOSHIBA for any infringement of patents or any other intellectual property rights of third parties that may result from the use of Product. No license to any intellectual property right is granted by this document, whether express or implied, by estoppel or otherwise.
- **ABSENT A WRITTEN SIGNED AGREEMENT, EXCEPT AS PROVIDED IN THE RELEVANT TERMS AND CONDITIONS OF SALE FOR PRODUCT, AND TO THE MAXIMUM EXTENT ALLOWABLE BY LAW, TOSHIBA (1) ASSUMES NO LIABILITY WHATSOEVER, INCLUDING WITHOUT LIMITATION, INDIRECT, CONSEQUENTIAL, SPECIAL, OR INCIDENTAL DAMAGES OR LOSS, INCLUDING WITHOUT LIMITATION, LOSS OF PROFITS, LOSS OF OPPORTUNITIES, BUSINESS INTERRUPTION AND LOSS OF DATA, AND (2) DISCLAIMS ANY AND ALL EXPRESS OR IMPLIED WARRANTIES AND CONDITIONS RELATED TO SALE, USE OF PRODUCT, OR INFORMATION, INCLUDING WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, ACCURACY OF INFORMATION, OR NONINFRINGEMENT.**
- Do not use or otherwise make available Product or related software or technology for any military purposes, including without limitation, for the design, development, use, stockpiling or manufacturing of nuclear, chemical, or biological weapons or missile technology products (mass destruction weapons). Product and related software and technology may be controlled under the applicable export laws and regulations including, without limitation, the Japanese Foreign Exchange and Foreign Trade Law and the U.S. Export Administration Regulations. Export and re-export of Product or related software or technology are strictly prohibited except in compliance with all applicable export laws and regulations.
- Please contact your TOSHIBA sales representative for details as to environmental matters such as the RoHS compatibility of Product. Please use Product in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. **TOSHIBA ASSUMES NO LIABILITY FOR DAMAGES OR LOSSES OCCURRING AS A RESULT OF NONCOMPLIANCE WITH APPLICABLE LAWS AND REGULATIONS.**