RA Family
IEC 60730/60335 Self Test Library for RA MCU (RA0_CM23)

R01AN7509EJ0100
Rev.1.00
Aug.25. 2025

Introduction

Today, as automatic electronic controls systems continue to expand into many diverse applications, the requirement of reliability and safety are becoming an ever increasing factor in system design.

For example, the introduction of the IEC60730 safety standard for household appliances requires manufactures to design automatic electronic controls that ensure safe and reliable operation of their products.

The IEC60730 standard covers all aspects of product design but Annex H is of key importance for design of Microcontroller based control systems. This provides three software classifications for automatic electronic controls:

1. Class A: Control functions, which are not intended to be relied upon for the safety of the equipment.
   Examples: Room thermostats, humidity controls, lighting controls, timers, and switches.
2. Class B: Control functions, which are intended to prevent unsafe operation of the controlled equipment.
   Examples: Thermal cut-offs and door locks for laundry equipment.
3. Class C: Control functions, which are intended to prevent special hazards
   Examples: Automatic burner controls and thermal cut-outs for closed.

Appliances such as washing machines, dishwashers, dryers, refrigerators, freezers, and Cookers/Stoves will tend to fall under the classification of Class B.

This Application Note provides guidelines of how to use flexible sample software routines to assist with compliance with IEC60730 class B safety standards. These routines have been certified by VDE Test and Certification Institute GmbH and a copy of the Test Certificate is available in the download package for this Application Note.

The software routines provided are to be used after reset and also during the program execution. This document and the accompanying sample code provide an example of how to do this.

Target

- Device:
  - Renesas RA Family (Arm® Cortex®-M23) * See next page for series and groups.
- Development environment:
  - GCC(GNU) Arm Embedded 13.2.1.arm-13-7 / e2 studio 2024-07 (24.7.0)

The term "RA MCU" used in this document refers to the following products.

Table 1. RA MCU Self-Test Function List

CPU Core		Arm® Cortex®-M23
Series		RA0
Group		RA0E1
Test Function	CPU	○(*1)
	ROM	○
	RAM	○
	Clock	○
	Independent Watchdog Timer (IWDT)	○
	ADC12	○
	GPIO	○

*1: Excluding FPU tests due to not implementing FPU.

Table of Contents

1. Tests

1.1 CPU

This section describes CPU tests routines. (Reference: *IEC 60730-1:2013+A1:2015+A2:2020 Annex H - Table H.11.12.7 1.CPU*)

This software tests the following CPU registers.

Table 1.1. List of Registers to Be Tested

CPU Core		Arm® Cortex®-M23
Group		RA0E1
Register to be Tested	General-purpose Register	R0-R12
	Control Register (*1)	MSP, PSP, LR, PSR(APSR, IPSR, EPSR), CONTROL
	Program Counter	PC

Notes: 1. Even if the register names are the same, the bit field configuration may differ depending on the device.

The source file cpu_test.c provides implementation of the CPU test using C language and relies on assembly language function to access the registers (that is, CPU_Test_Control). File cpu_test_coupling.c is also required to use the coupling test of the General Purpose Registers. Coupling test relies on assembly language functions:

- TestGPRsCouplingStart_A
- TestGPRsCouplingR1_R3_A
- TestGPRsCouplingR4_R6_A
- TestGPRsCouplingR7_R9_A
- TestGPRsCouplingR10_R12_A
- TestGPRsCouplingR0_A
- TestGPRsCouplingStart_B
- TestGPRsCouplingR1_R3_B
- TestGPRsCouplingR4_R6_B
- TestGPRsCouplingR7_R9_B
- TestGPRsCouplingR10_R12_B
- TestGPRsCouplingR0_B
- TestGPRsCouplingEnd

Alternatively, CPU_Test_General_Low, CPU_Test_General_High assembly language functions are used to test General-purpose registers.

The header file cpu_test.h header file provides the interface to the CPU tests.

These tests are testing such fundamental aspects of the CPU operation; the API functions do not have return values to indicate the result of a test. Instead the user of these tests must create an error handling function with the following declaration:

```
extern void CPU_Test_ErrorHandler(void);
```

The CPU test will jump to this function if an error is detected. This function must not return.

All the test functions follow the rules of register preservation following a C function call. Therefore the user can call these functions like any normal C function without any additional responsibilities for saving register values beforehand.

1.1.1 CPU Test Software API

Table 1.2. CPU Test Software API Source Files

File Name
cpu_test.h
cpu_test.c cpu_test_coupling.c
TestGPRsCouplingStart_A.asm TestGPRsCouplingR0_A.asm TestGPRsCouplingR1_R3_A.asm TestGPRsCouplingR4_R6_A.asm TestGPRsCouplingR7_R9_A.asm TestGPRsCouplingR10_R12_A.asm TestGPRsCouplingStart_B.asm TestGPRsCouplingR0_B.asm TestGPRsCouplingR1_R3_B.asm TestGPRsCouplingR4_R6_B.asm TestGPRsCouplingR7_R9_B.asm TestGPRsCouplingR10_R12_B.asm TestGPRsCouplingEnd.asm CPU_Test_Control.asm CPU_Test_General_Low.asm CPU_Test_General_High.asm

■ cpu_test.c File

Syntax	
void CPU_Test_All(void)	
Description	
Run through all the tests detailed below in the following order: 1. If using Coupling General Purpose Registers tests (*1. see below): CPU_Test_GPRsCouplingPartA CPU_Test_GPRsCouplingPartB 2. If not using Coupling General Purpose Registers test: CPU_Test_General_Low CPU_Test_General_High 3. CPU_Test_Control 4. CPU_Test_PC It is the calling function's responsibility to ensure that the processor is in Privileged Mode. If this function is called in unprivileged mode, the test will fail as some of the register bits are not accessible in unprivileged mode. In addition, since the CPU_Test_Control function tests stack pointer registers (that is, MSP and PSP), it is also the calling function's responsibility to ensure no interrupts occur during this test. If an error is detected, external function CPU_Test_ErrorHandler will be called. See the individual tests for a full description. Note: A #define USE_TEST_GPRS_COUPLING in the code is used to select which functions will be used to test the General-Purpose Registers.	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

Syntax	
void CPU_Test_PC(void)	
Description	
This function tests the Program Counter (PC) register. Test that the PC is operating by calling a separate function. Call a separate function (TestPC_TestFunction) that returns the inverted value of the specified parameter and check the return value. This confirms that the PC is working properly. If an error is detected, external function CPU_Test_ErrorHandler will be called.	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

■ CPU_Test_General_Low.asm File

Syntax	
void CPU_Test_General_Low(void)	
Description	
Tests general-purpose registers R0, R1, R2, R3, R4, R5, R6 and R7. Registers are tested in pairs. For each pair of registers: 1. Write h'55555555 to both. 2. Read both and check they are equal. 3. Write h'AAAAAAAA to both. 4. Read both and check they are equal. It is the calling function’s responsibility to disable exception during this test. If an error is detected, external function CPU_Test_ErrorHandler will be called.	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

■ CPU_Test_General_High.asm File

Syntax	
void CPU_Test_General_High(void)	
Description	
Tests general-purpose registers R8, R9, R10, R11 and R12. These are the general-purpose registers. Registers are tested in pairs. For each pair of registers: 1. Write h'55555555 to both. 2. Read both and check they are equal. 3. Write h'AAAAAAAA to both. 4. Read both and check they are equal. It is the calling function’s responsibility to disable exceptions during this test. If an error is detected, external function CPU_Test_ErrorHandler will be called.	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

■ CPU_Test_Control.asm File

Syntax	
void CPU_Test_Control(void)	
Description	
Tests control registers (Refer to "Table 1.1. List of Registers to Be Tested" because it depends on the device). This test assumes registers R1 to R4 are working. Generally, the test procedure for each register is as follows. For each register: 1. Write h'55555555 to. 2. Read back and check value equals h'55555555. 3. Write h'AAAAAAAA to. 4. Read back and check value equals h'AAAAAAAA. However, note that there are some cases where restrictions on specific bits within a register may not allow this procedure. For these cases other test values have been chosen. It is the calling function’s responsibility to ensure that the processor is in Privileged Mode. If this function is called in Unprivileged Mode the test will fail as some of the register bits are not accessible in Unprivileged Mode. It is also the calling function’s responsibility to disable exceptions during this test. Note: FAULTMASK and PRIMASK are not tested since this test requires exceptions be disabled. Thus, they are not activated during the test modifying FAULTMASK and PRIMASK. If an error is detected, external function CPU_Test_ErrorHandler will be called.	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

■ cpu_test_coupling.c File

Syntax	
void CPU_Test_GPRsCouplingPartA(void)	
Description	
Test general-purpose registers R0 to R12. Coupling faults between the registers are detected. The general-purpose register test consists of Part A and Part B, and this function is Part A. It is the calling function's responsibility to ensure no interrupts occur during this test. If an error is detected, external function CPU_Test_ErrorHandler will be called.	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

Syntax	
void CPU_Test_GPRsCouplingPartB(void)	
Description	
Tests general-purpose registers R0 to R12. Coupling faults between the registers are detected. The general-purpose register test consists of Part A and Part B, and this function is Part B. It is the calling function's responsibility to ensure no interrupts occur during this test. If an error is detected, external function CPU_Test_ErrorHandler will be called.	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

1.2 ROM

This section describes the ROM/Flash memory test using CRC routines. (Reference: *IEC 60730-1:2013+A1:2015+A2:2020 Annex H– H2.19.4.2 CRC – Double Word*)

CRC is a fault/error control technique which generates a single word or checksum to represent the contents of memory.

A CRC checksum is the remainder of a binary division with no bit carry (XOR is used instead of subtraction) of the message bit stream, by a predefined (short) bit stream of length n + 1. which represents the coefficients of a polynomial with degree n. Before the division, n zeros are appended to the message stream. CRCs are often used because they are simple to implement in binary hardware and are easy to analyze mathematically.

The ROM test can be achieved by generating a CRC value for the contents of the ROM and saving it. During the memory self-test, the same CRC algorithm is used to generate another CRC value, which is compared with the saved CRC value. The technique recognizes all one-bit errors and a high percentage of multi-bit errors.

The complicated part of using CRCs is if you need to generate a CRC value that will then be compared with other CRC values produced by other CRC generators. This proves difficult because there are a number of factors that can change the resulting CRC value even if the basic CRC algorithm is the same. This includes the combination of the order that the data is supplied to the algorithm, the assumed bit order in any look-up table used and the required order of the bits of the actual CRC value. This complication has arisen because big- and little-endian systems were developed to work together that employed serial data transfers where bit order became important. Also, some debuggers implement a software break on ROM, in which case the contents of ROM may be rewritten during debugging.

The method of calculating the reference CRC value depends on the toolchain used. For the detailed procedure, refer to Section 2.2 ROM in 2.Example Usage

1.2.1 CRC32 Algorithm

The RA MCU includes a CRC module that includes support for CRC32. This software sets the CRC module to produce a 32-bit CRC32.

- Polynomial = 0x04C11DB7 ($x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$)
- Width = 32 bits
- Initial value = 0xFFFFFFFF
- XOR with h'FFFFFFFF is performed on the output CRC

1.2.2 ROM Test Software API

The functions in the reminder of this section are used to calculate a CRC value and verify its correctness against a value stored in ROM.

All software is written in ANSI C. The renesas.h header file includes definition of RA MCU registers.

Table 1.3. ROM Test Software API Source Files

File Name
crc.h crc_verify.h
crc.c CRC_Verify.c

■ CRC_Verify.c File

Syntax	
bool_t CRC_Verify(const uint32_t ui32_NewCRCValue, const uint32_t ui32_AddrRefCRC)	
Description	
This function compares a new CRC value with a reference CRC by supplying address where reference CRC is stored.	
Input Parameters	
const uint32_t ui32_NewCRCValue	Value of calculated new CRC value.
const uint32_t ui32_AddrRefCRC	Address where 32 bit reference CRC value is stored.
Output Parameters	
NONE	N/A
Return Values	
bool_t	True = Passed, False = Failed

■ crc.c File

Syntax	
void CRC_Init(void)	
Description	
Initializes the CRC module. This function must be called before any of the other CRC functions can be.	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

Syntax	
uint32_t CRC_Calculate(const uint32_t* pui32_Data, uint32_t ui32_Length)	
Description	
This function calculates the CRC of a single specified memory area.	
Input Parameters	
const uint32_t* pui32_Data	Pointer to start of memory to be tested.
uint32_t ui32_Length	Length of the data in long words.
Output Parameters	
NONE	N/A
Return Values	
Uint32_t	The 32-bit calculated CRC32 value.

The following functions are used when the memory area cannot simply be specified by a start address and length. They provide a way of adding memory areas in ranges/sections. This can also be used if function CRC_Calculate takes too long in a single function call.

■ crc.c File

Syntax	
void CRC_Start(void)	
Description	
Resets the CRC module to the start condition. Call this once prior to using function CRC_AddRange.	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

Syntax	
void CRC_AddRange(const uint32_t* pui32_Data, uint32_t ui32_Length)	
Description	
Use this function rather than CRC_Calculate to calculate the CRC on data made up of more than one address range. Call CRC_Start first, then call CRC_AddRange for each address range required and call CRC_Result to get the CRC value.	
Input Parameters	
const uint32_t* pui32_Data	Pointer to start of memory range to be tested.
uint32_t ui32_Length	Length of the data in long words.
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

Syntax	
uint32_t CRC_Result(void)	
Description	
Returns the bit-reversed value of the value read from the CRC data output register (CRCDOR) as the return value.	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
uint32_t	The calculated CRC32 value.

1.3 RAM

March tests are a family of tests that are well recognized as an effective way of testing RAM.

A March test consists of a finite sequence of March elements. A March element is a finite sequence of operations applied to every cell in the memory array before proceeding to the next cell.

In general, the more March elements the algorithm consists of, the better its fault coverage will be but at the expense of a slower execution time.

The algorithms themselves are destructive (they do not preserve the current RAM values) but the supplied test functions provide a non-destructive option so that memory contents can be preserved. This is achieved by copying the memory to a supplied buffer before running the actual algorithm and then restoring the memory from the buffer at the end of the test. The API includes an option for automatically testing the buffer as well as the RAM test area.

The area of RAM being tested cannot be used for anything else while it is being tested. This makes the testing of RAM used for the stack particularly difficult. To help with this problem the API includes functions which can be used for testing the stack.

The following section introduces the specific March Tests. Following that is the specification of the software APIs.

1.3.1 RAM Test Algorithms

(1) March C-

The March C algorithm (van de Goor 1991) consists of 6 March elements with a total of 10 operations. It detects the following faults.

1. Stuck-At Faults (SAF)
   - The logic value of a cell or a line is always 0 or 1.
2. Transition Faults (TF)
   - A cell or a line that fails to undergo a 0→1 or a 1→0 transition.
3. Coupling Faults (CF)
   - A write operation to one cell changes the content of a second cell.
4. Address Decoder Faults (AF)
   - Any fault that affects the address decoder
   - With a certain address, no cell will be accessed.
   - A certain cell is never accessed.
   - With a certain address, multiple cells are accessed simultaneously.
   - A certain cell can be accessed by multiple addresses.

These are the 6 March elements.

1. Write all zeros to array.
2. Starting at lowest address, read zeros, write ones, increment up array bit by bit.
3. Starting at lowest address, read ones, write zeros, increment up array bit by bit.
4. Starting at highest address, read zeros, write ones, decrement down array bit by bit.
5. Starting at highest address, read ones, write zeros, decrement down array bit by bit.
6. Read all zeros from array.

(2) March X

Note: This algorithm has not been implemented for the RA MCU and is only presented here for information as it relates to the March X WOM algorithm below.

The March X algorithm consists of 4 March elements with a total of 6 operations. It detects the following faults.

1. Stuck-At Faults (SAF)
2. Transition Faults (TF)
3. Inversion Coupling Faults (CFin)
4. Address Decoder Faults (AF)

These are the 4 March elements.

1. Write all zeros to array.
2. Starting at lowest address, read zeros, write ones, increment up array bit by bit.
3. Starting at highest address, read ones, write zeros, decrement down array bit by bit.
4. Read all zeros from array.

(3) March X WOM(Word-Oriented Memory Version)

The March X Word-Oriented Memory (WOM) algorithm has been created from a standard March X algorithm in two stages. First, the standard March X is converted from using a single-bit data pattern to using a data pattern equal to the memory access width. At this stage the test is primarily detecting inter-word faults including Address Decoder faults. The second stage is to add an additional two March elements. The first uses a data pattern of alternating high/low bits then the second uses the inverse. The addition of these elements is to detect intra-word coupling faults.

These are the 6 March elements.

1. Write all zeros to array.
2. Starting at lowest address, read zeros, write ones, increment up array word by word.
3. Starting at highest address, read ones, write zeros, decrement down word by word.
4. Starting at lowest address, read zeros, write h’AAs, increment up array word by word.
5. Starting at highest address, read h’AAs, write h’55s, decrement down word by word.
6. Read all h’55s from array.

1.3.2 RAM Test Software API

APIs are prepared corresponding to two test algorithms in the RAM Test.

Each API will be explained.

(1) March C- algorithm Software API

This test can be configured to use 8-, 16- or 32-bit RAM accesses.

This is achieved by #defining RAMTEST_MARCH_C_ACCESS_SIZE in the header file to be one of the following.

1. RAMTEST_MARCH_C_ACCESS_SIZE_8BIT
2. RAMTEST_MARCH_C_ACCESS_SIZE_16BIT
3. RAMTEST_MARCH_C_ACCESS_SIZE_32BIT

Sometimes limiting the maximum size of RAM that can be tested with a single function call can speed the test up as well as reducing stack and code size. This is done by limiting the size of the variable used to hold the number of ‘words’ that the test area contains. The ‘word’ size is the selected access width.

This is achieved by #defining RAMTEST_MARCH_C_MAX_WORDS in the header file to be one of the following.

1. RAMTEST_MARCH_C_MAX_WORDS_8BIT (Max words in test area is 0xFF)
2. RAMTEST_MARCH_C_MAX_WORDS_16BIT (Max words in test area is 0xFFFF)
3. RAMTEST_MARCH_C_MAX_WORDS_32BIT (Max words in test area is 0xFFFFFFFF)

Table 1.4. March C- Algorithm Software API Source Files

File name
ramtest_march_c.h
ramtest_march_c.c

The source is written in ANSI C and uses renesas.h header file to access peripheral registers.

Note: The API allows just a single word to be tested with a function call. However, for coupling faults to be tested between words, it is important to use the functions to test a data range bigger than one word.

■ ramtest_march_c.c File

Syntax	
bool_t RamTest_March_C(const uint32_t ui32_StartAddr, const uint32_t ui32_EndAddr, void* const p_RAMSafe)	
Description	
RAM memory test using March C (Goor 1991) algorithm.	
Input Parameters	
const uint32_t ui32_StartAddr	The address of the first word of RAM to be tested. This must be aligned with the selected memory access width.
const uint32_t ui32_EndAddr	The address of the last word of RAM to be tested. This must be aligned with the selected memory access width and be a value greater or equal to ui32_StartAddr.
void* const p_RAMSafe	For a destructive memory test, set to NULL. For a non-destructive memory test, set to the start of a buffer that is large enough to copy the contents of the test area into it and that is aligned with the selected memory access width.
Output Parameters	
NONE	N/A
Return Values	
bool_t	True = Test passed, False = Test or parameter check failed.

Syntax	
bool_t RamTest_March_C_Extra(const uint32_t ui32_StartAddr, const uint32_t ui32_EndAddr, void* const p_RAMSafe)	
Description	
Non-Destructive RAM memory test using March C (Goor 1991) algorithm. This function differs from the RamTest_March_C function by testing the ‘RAMSafe’ buffer before using it. If the test of the ‘RAMSafe’ buffer fails, the test will be aborted, and the function will return false.	
Input Parameters	
const uint32_t ui32_StartAddr	The address of the first word of RAM to be tested. This must be aligned with the selected memory access width.
const uint32_t ui32_EndAddr	The address of the last word of RAM to be tested. This must be aligned with the selected memory access width and be a value greater or equal to ui32_StartAddr.
void* const p_RAMSafe	Set to the start of a buffer that is large enough to copy the contents of the test area into it and that is aligned with the selected memory access width.
Output Parameters	
NONE	N/A
Return Values	
bool_t	True = Test passed, False = Test or parameter check failed.

(2) March X WOM algorithm Software API

This test can be configured to use 8-, 16- or 32-bit RAM accesses.

This is achieved by #defining RAMTEST_MARCH_X_WOM_ACCESS_SIZE in the header file to be one of the following.

- RAMTEST_MARCH_X_WOM_ACCESS_SIZE_8BIT
- RAMTEST_MARCH_X_WOM_ACCESS_SIZE_16BIT
- RAMTEST_MARCH_X_WOM_ACCESS_SIZE_32BIT

In order to speed up the run time of the test you can choose to limit the maximum size of RAM that can be tested with a single function call. This is done by limiting the size of the variable used to hold the number of 'words' that the test area contains. The 'word' size is the same as the selected access width.

This is achieved by #defining RAMTEST_MARCH_X_WOM_MAX_WORDS in the header file to be one of the following.

- RAMTEST_MARCH_X_WOM_MAX_WORDS_8BIT (Max words in test area is 0xFF)
- RAMTEST_MARCH_X_WOM_MAX_WORDS_16BIT (Max words in test area is 0xFFFF)
- RAMTEST_MARCH_X_WOM_MAX_WORDS_32BIT (Max words in test area is 0xFFFFFFFF)

Table 1.5. March X WOM algorithm Software API Source Files

File name
ramtest_march_x_wom.h
ramtest_march_x_wom.c

The source is written in ANSI C and uses renesas.h header file to access peripheral registers.

Note: The API allows just a single word to be tested with a function call. However, for coupling faults to be tested between words it is important to use the functions to test a data range bigger than one word.

■ ramtest_march_x_wom.c File

Syntax	
bool_t RamTest_March_X_WOM(const uint32_t ui32_StartAddr, const uint32_t ui32_EndAddr, void* const p_RAMSafe)	
Description	
RAM memory test based on March X algorithm converted for WOM.	
Input Parameters	
const uint32_t ui32_StartAddr	Address of the first word of RAM to be tested. This must be aligned with the selected memory access width.
const uint32_t ui32_EndAddr	Address of the last word of RAM to be tested. This must be aligned with the selected memory access width and be a value greater or equal to ui32_StartAddr.
void* const p_RAMSafe	For a destructive memory test, set to NULL. For a non-destructive memory test, set to the start of a buffer that is large enough to copy the contents of the test area into it and that is aligned with the selected memory access width
Output Parameters	
NONE	N/A
Return Values	
bool_t	True = Test passed, False = Test or parameter check failed.

Syntax	
bool_t RamTest_March_X_WOM_Extra(const uint32_t ui32_StartAddr, const uint32_t ui32_EndAddr, void* const p_RAMSafe)	
Description	
Non-Destructive RAM memory test based on March X algorithm converted for WOM. This function differs from the RamTest_March_X_WOM function by testing the ‘RAMSafe’ buffer before using it. If the test of the ‘RAMSafe’ buffer fails, then the test will be aborted, and the function will return false.	
Input Parameters	
const uint32_t ui32_StartAddr	The address of the first word of RAM to be tested. This must be aligned with the selected memory access width.
const uint32_t ui32_EndAddr	The address of the last word of RAM to be tested. This must be aligned with the selected memory access width and be a value greater or equal to ui32_StartAddr.
void* const p_RAMSafe	Set to the start of a buffer that is large enough to copy the contents of the test area into it and that is aligned with the selected memory access width.
Output Parameters	
NONE	N/A
Return Values	
bool_t	True = Test passed, False = Test or parameter check failed.

(3) RAM Test(Stack) Software API

This API enables a RAM test to be performed on an area of RAM that includes the stack. As the function that performs the RAM test requires a stack these functions will, re-locate the stack to a supplied new RAM area allowing the original stack area to be tested. Three functions are provided that can be called depending upon which stack (Main or Process) is in the test area or if both are.

It is the calling function's responsibility to ensure that the processor is in Privileged Mode. If this function is called in unprivileged mode the test will fail as some of the register bits are not accessible in unprivileged mode.

Note: The stack testing functions make use of one of the March RAM tests presented previously by passing it in as a function pointer. If using a test that requires initialization before use it is the user's responsibility to ensure this has been done before trying to use the test by calling one of these functions.

Table 1.6. RAM Test(Stack) Software API Source Files

File name
ramtest_stack.h
ramtest_stack.c
StartBothTestAssembly.asm StartMainTestAssembly.asm StartProcTestAssembly.asm

■ ramtest_stack.c File

Syntax	
bool_t RamTest_Stack_Main(	const uint32_t ui32_StartAddr, const uint32_t ui32_EndAddr, void* const p_RAMSafe, const uint32_t ui32_NewMSP, const TEST_FUNC fpTest_Func)
Description	
RAM test of an area that includes the Main Stack (but not includes the Process stack).	
Input Parameters	
const uint32_t ui32_StartAddr	The address of the first word of RAM to be tested. This must be compatible with the requirements of fpTest_Func.
const uint32_t ui32_EndAddr	The address of the last word of RAM to be tested. This must be compatible with the requirements of fpTest_Func.
void* const p_RAMSafe	Set to the start of a buffer that is the same size as the test RAM area. This must be compatible with the requirements of fpTest_Func.
const uint32_t ui32_NewUSP	New Stack pointer value for the Main stack to be relocated to.
const TEST_FUNC fpTest_Func	Function pointer of type TEST_FUNC to the actual memory test to be used. Typedef bool_t(*TEST_FUNC)(uint32_t, uint32_t, void*); For example, 'RamTest_March_X_WOM'.
Output Parameters	
NONE	N/A
Return Values	
bool_t	True = Test passed, False = Test or parameter check failed.

Syntax	
bool_t RamTest_Stack_Proc(	const uint32_t ui32_StartAddr, const uint32_t ui32_EndAddr, void* const p_RAMSafe, const uint32_t ui32_NewPSP, const TEST_FUNC fpTest_Func)
Description	
RAM test of an area that includes the Process stack (but not includes the Main stack).	
Input Parameters	
const uint32_t ui32_StartAddr	The address of the first word of RAM to be tested. This must be compatible with the requirements of the fpTest_Func.
const uint32_t ui32_EndAddr	The address of the last word of RAM to be tested. This must be compatible with the requirements of the fpTest_Func.
void* const p_RAMSafe	Set to the start of a buffer that is the same size as the test RAM area. This must be compatible with the requirements of the fpTest_Func.
const uint32_t ui32_NewPSP	New Stack pointer value for the Process stack to be relocated to.
const fpTest_Func	Function pointer of type TEST_FUNC to the actual memory test to be used. Typedef bool_t(*TEST_FUNC)(uint32_t, uint32_t, void*); For example, ‘RamTest_March_X_WOM’.
Output Parameters	
NONE	N/A
Return Values	
bool_t	True = Test passed, False = Test or parameter check failed.

Syntax	
bool_t RamTest_Stacks(	const uint32_t ui32_StartAddr, const uint32_t ui32_EndAddr, void* const p_RAMSafe, const uint32_t ui32_NewPSP, const uint32_t ui32_NewMSP, const TEST_FUNC fpTest_Func)
Description	
RAM test of an area that includes both the Main stack and the Process stack.	
Input Parameters	
const uint32_t ui32_StartAddr	The address of the first word of RAM to be tested. This must be compatible with the requirements of the fpTest_Func.
const uint32_t ui32_EndAddr	The address of the last word of RAM to be tested. This must be compatible with the requirements of the fpTest_Func.
void* const p_RAMSafe	Set to the start of a buffer that is the same size as the test RAM area. This must be compatible with the requirements of the fpTest_Func.
const uint32_t ui32_NewPSP	New Stack pointer value for the Process stack to be relocated to.
const uint32_t ui32_NewMSP	New Stack pointer value for the Main stack to be relocated to.
const TEST_FUNC fpTest_Func	Function pointer of type TEST_FUNC to the actual memory test to be used. Typedef bool_t(*TEST_FUNC)(const uint32_t, const uint32_t, void* const); For example, RamTest_March_X_WOM
Output Parameters	
NONE	N/A
Return Values	
bool_t	True = Test passed, False = Test or parameter check failed.

1.4 Clock

This test verifies that the system clock frequency is within the allowable range by using the input pulse interval measurement function of the MCU's built-in timer array unit (TAU) channel 5. If it is outside the range, a frequency anomaly is detected and error processing is performed.

1.4.1 System Clock Frequency Monitoring by "TAU0 ch5 (input pulse interval measurement function)"

The count starts at the active edge of the input pulse signal selected by the timer input selection register 0 (TIS0.TIS[2:0]). The timer count value is captured at the active edge of the next pulse. In this way, the input pulse interval is measured.

Select "LOCO" as the timer input selected by the timer input selection register 0 (TIS0.TIS[2:0]), and select "PCLKB (no division)" as the operating clock for channel 5 in the related register (*1).

Note: 1. Set TMR05.CCS=0, TMR05.CKS[1:0]=b'00, TPS0.PRS0[3:0]=0x0

In RA0E1, PCLKB (peripheral module clock) is the same clock source as the system clock (ICLK).

Below is an internal block diagram of channel 5 of the timer array

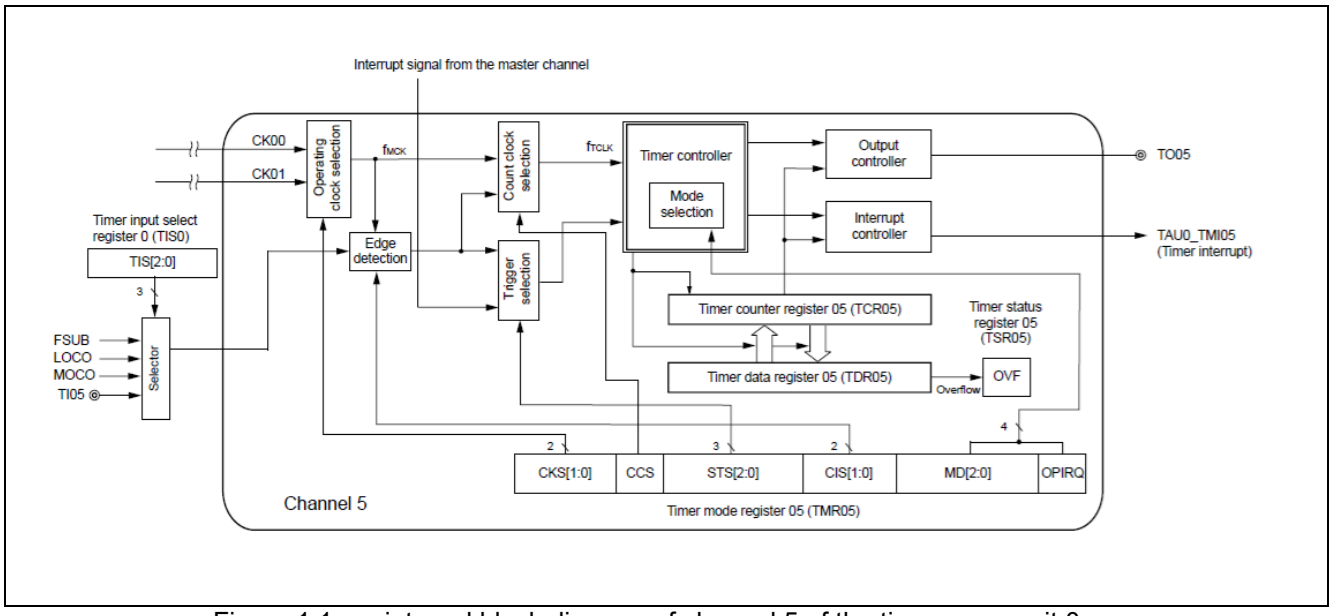

Figure 1.1 an internal block diagram of channel 5 of the timer array unit 0

In this sample software, when a valid edge of the LOCO input is detected, it starts counting using PCLKB as the operating clock, and when the next valid edge is detected, the value of the TCR05 register is captured in the Timer Data Register 05 (TDR05) and TAU0_TMI05 is generated. (The TCR05 register starts counting again from 0x0000.)

* For details, refer to the "Timer Array Unit (TAU)" chapter of the User's Manual : Hardware.

It also checks within the TAU0 ch5 interrupt whether the system clock frequency is within the set range, and executes error handling if it is out of range.

See Chapter 2.4 for an example of how to use this test module.

1.4.2 Clock Test Software API

Table 1.7. Clock Test Software API Source Files

File Name
clock_monitor.h
clock_monitor.c

The test module relies on the renesas.h header file to access to peripheral registers.

■ clock_monitor.c File

Syntax	
void ClockMonitor_Init(CLOCK_MONITOR_ERROR_CALL_BACK CallBack)	
Description	
1. Register the error detection processing function. 2. Initialize the TAU0 (ch5) module and start monitoring PCLKB(system clock is the clock source). 3. If LOCO is stopped, start oscillation.	
Input Parameters	
CLOCK_MONITOR_ERROR_CALL_BACK CallBack	A function that is called when PCLKB clock is out of tolerance.
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

Syntax	
void tau0_tmi05_int (void)	
Description	
TAU0 ch5 capture completion interrupt handler. Determines whether the captured input pulse is within the valid range, and if it is outside the valid range, calls the callback function registered in the ClockMonitor_Init function. * The valid range is defined in clock_monitor.h as “hwMAXTIME” and “hwMINTIME”.	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

1.5 Independent Watchdog Timer (IWDT)

A watchdog timer is used to detect abnormal program execution. If a program is not running as expected, the watchdog timer will not be refreshed by software as required and will therefore detect an error.

The Independent Watchdog Timer (IWDT) module of the RA MCU is used for this. It includes a windowing feature so that the refresh must happen within a specified ‘window’ rather than just before a specified time. It can be configured to generate an internal reset or an NMI interrupt if an error is detected.

All the configurations for IWDT can be done through the Option Function Select Register 0 (OFS0) in Option-Setting Memory whose settings are controlled by the user (see Section 2.5 for an example of configuration). The option setting memory is a series of registers that can be used to select the state of the microcontroller after reset and is located in the code flash area.

The test module relies on the renesas.h header file to access to peripheral registers.

1.5.1 IWDT Software API

Table 1.8. IWDT Software API Source Files

File Name
iwdt.h
iwdt.c

■ iwdt.c File

Syntax	
void IWDT_Init (void)	
Description	
Initialize the independent watchdog timer. After calling this, the IWDT_Kick function must then be called at the correct time to prevent a watchdog timer error. Note: If configured to produce an interrupt then this will be the Non Maskable Interrupt (NMI). This must be handled by user code which must check the NMISR.IWDTST flag.	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

Syntax	
void IWDT_Kick(void)	
Description	
Refresh the watchdog timer count.	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

Syntax	
bool_t IWDT_DidReset(void)	
Description	
Returns true if the IWDT has timed out or not been refreshed correctly. In addition, the underflow flag and refresh error flag are cleared. * The default setting for Class-B is “No window control (100%)”.	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
bool_t	“True” if IWDT has timed out, or was not updated correctly. otherwise “False”.

1.6 ADC

This ADC (ADC12) has a function that allows you to select the + side reference voltage, - side reference voltage, or internal reference voltage for test A/D conversion.

This function can be used to check whether the A/D converter is operating correctly. This sample software assumes 12-bit resolution.

About the test mode setting for this ADC, refer to the "12-bit A/D Converter (ADC12)" chapter in User's Manual: Hardware.

Table 1.9. Built-in ADC

Group	RA0E1
ADC	ADC12
Number of units	1
Diagnostic reference voltage	The '-' side reference voltage / internal reference voltage / The '+' side reference voltage

Note: The relationship between each measurement mode and the diagnostic reference voltage is as follows (※This sample software)

When measuring zero scale select the "-" side reference voltage as the conversion target.

When measuring full scale, select the "+" side reference voltage as the conversion target.

The test module relies on the renesas.h header file to access to peripheral registers.

1.6.1 ADC Test Software API

Table 1.10. ADC Test Software API Source Files

File Name
test_adc.h
test_adc.c

■ test_adc.c File

Syntax	
void Test_ADC_Init(void)	
Description	
Initialize the ADC module (Unit 0). This function must be called before using any other ADC functions.	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

Syntax	
bool_t Test_ADC_Wait(void)	
Description	
This function performs basic settings for the ADC module (unit 0), switches the target for A/D conversion for testing, waits while A/D conversion is in progress. It determines whether the A/D conversion result is within the acceptable range, and if it is within the range, it returns TRUE (1), and if it is outside the range, it determines that there is an error and returns FALSE (0). Note that this function does not move to the error handling function. ※ Sets the ADM0, ADM1, ADM2, ADUL, ADLL, and ADTES registers.	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
bool_t	True = Test passed, False = Test failed.

Syntax	
void Test_ADC_Start(const ADC_ERROR_CALL_BACK Callback)	
Description	
・Switch the A/D conversion target for testing the ADC module (unit 0) and perform A/D conversion. * Since the first A/D conversion data must be ignored depending on the start timing, this function performs two A/D conversion operations. ・After the A/D conversion is complete, call the Test_ADC_CheckResult function to check the A/D conversion results. * Set the ADM0, ADUL, ADLL, and ADTES registers.	
Input Parameters	
const ADC_ERROR_CALL_BACK Callback	Function to call if an error is detected. Note: This function will only get called if **Test_ADC_CheckResult** is called with parameter **bCallErrorHandler** set **true**. This function only stores the callback content in **g_ADC_ErrorCallBack**.
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

Syntax	
bool_t Test_ADC_CheckResult(const bool_t bCallErrorHandler)	
Description	
・Verify that the test A/D conversion results of the ADC module (unit 0) are within the acceptable range (note) for each target. ・Call this function after the **Test_ADC_Start** function, and call it each time the test A/D conversion is completed. • If the A/D conversion result is within the range, it returns **TRUE (1)**. • If the A/D conversion result is outside the range, it is judged as an error and returns **FALSE (0)**. Note that if the argument "**bCallErrorHandler**" is "**True**," it is judged as an error and moves to the error handling function. Note: For the acceptable range, refer to **test_adc.h**.	
Input Parameters	
const bool_t bCallErrorHandler	Set **True** to call the error call-back function supplied to function **Test_ADC_Start**, otherwise **False**.
Output Parameters	
NONE	N/A
Return Values	
bool_t	True = Test passed, False = Test failed.

1.7 GPIO

The GPIO readback level detection function is a function to read the digital output level of a pin when the port is in output mode. This makes it possible to diagnose defects in the terminals.

The GPIO readback level detection function checks the pins in the following procedure.

1. As preprocessing, set the PMC bit (terminal mode control) of the relevant terminal to “0” (digital input/output).
   * It is required for RA0E1
2. Set the PDRn (n=00-15) bits of the Pmn direction register (PDRm: m=0-9) to “1” and set them as output ports.
3. Set “1” in the POSRn (n=00-15) bit of the Pmn output setting register (POSRm: m=0-9) to output High.
4. Read the pin status from PIDRn (n=00 to 15) of the Pmn status register (PIDRm: m=0 to 9) and check for a high level.
5. Set “1” in the PORRn (n=00-15) bit of the Pmn output reset register (PORRm: m=0-9) to output Low.
6. Read the pin status from PIDRn (n=00 to 15) of the Pmn status register (PIDRm: m=0 to 9) and check for a low level.

The port to be tested is specified in the gpio_config.h header file.

1.7.1 GPIO Test Software API

Table 1.11. GPIO Test Software API Source File

File Name
gpio.h gpio_config.h
gpio.c

- gpio.c File

Syntax	
void GPIO_Start(const GPIO_ERROR_CALL_BACK Callback)	
Description	
This function uses the GPIO readback level detection function to switch the digital output level of the pin when the port is in output mode, and then reads it to diagnose pins defects. The port to be tested is specified in the gpio_config.h header file.	
Input Parameters	
const GPIO_ERROR_CALL_BACK Callback	A function to be called when a pin defect is detected (the read value of the port is different from the expected value)
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

2. Example Usage

This section gives to the user some useful suggestions about how to apply the released software.

Self test can be divided into two patterns:

(a) Power-on Test

These are tests run once after a reset. They should be run as soon as possible but if the startup time is especially important, it may be permissible to run some initialization code before running all the tests. For example, a faster main clock can be selected.

(b) Periodic Test

These are tests that are run regularly throughout normal program operation. This document does not provide a judgment of how often a particular test should be executed. The method for scheduling the periodic tests is determined by the user depending upon the structure of their application.

The following sections provide an example of how each test type should be used.

2.1 CPU

If a fault is detected by any of the CPU tests, a user supplied function called CPU_Test_ErrorHandler will be called. As any error in the CPU is very serious the aim of this function should be to get to a safe state, where software execution is not relied upon, as soon as possible.

2.1.1 Power-On

All the CPU tests should be run as soon as possible following a reset.

Note: The function must be called before the device is put in Unprivileged mode.
The function CPU_Test_All can be used to automatically run all the CPU tests.

2.1.2 Periodic

To test the CPU periodically, the function CPU_Test_All can be used, as it is for the power-on tests, to automatically run all CPU tests. Alternatively, to reduce the amount of testing done in a single function call, the user can choose to call each of the individual CPU test functions in turn each time the CPU periodic test is scheduled.

2.2 ROM

The ROM is tested by calculating a CRC value (CRC32) of its contents and comparing with a reference CRC value that must be added to a specific location in the ROM not included in the CRC calculation.

The CRC module must be initialized before use with a call of CRC_Init function.

Ensure that all ROM sections used are included in both the preliminary CRC calculation value and the ROM test so that the results will match.

2.2.1 Reference CRC Value Calculation in Advance

Since the GNU tool does not have a CRC calculation function, use the SRecord tool (*1) introduced below to calculate the reference CRC value. The user uses this tool to write the CRC value for reference in ROM in advance and compares it with this value in the self-test.

Note: SRecord is an open source project on SourceForge. See below for details.

- SRecord Web Site (SRecord v1.65)
  http://srecord.sourceforge.net/
- CRC Checksum Generation with “SRecord” Tools for GNU and Eclips
  https://llvm-gcc-renesas.com/wiki/index.php?title=CRC_Checksum_Generation_with_%E2%80%98SRecord%E2%80%99_Tools_for_GNU_and_Eclipse

When you unzip the downloaded file (srecord-1.65.0-win64.zip), the following programs will be extracted to the \srecord-1.65.0-win64\bin folder.

* The reference manual is included in the \srecord-1.65.0-win64\share\doc\srecord folder.

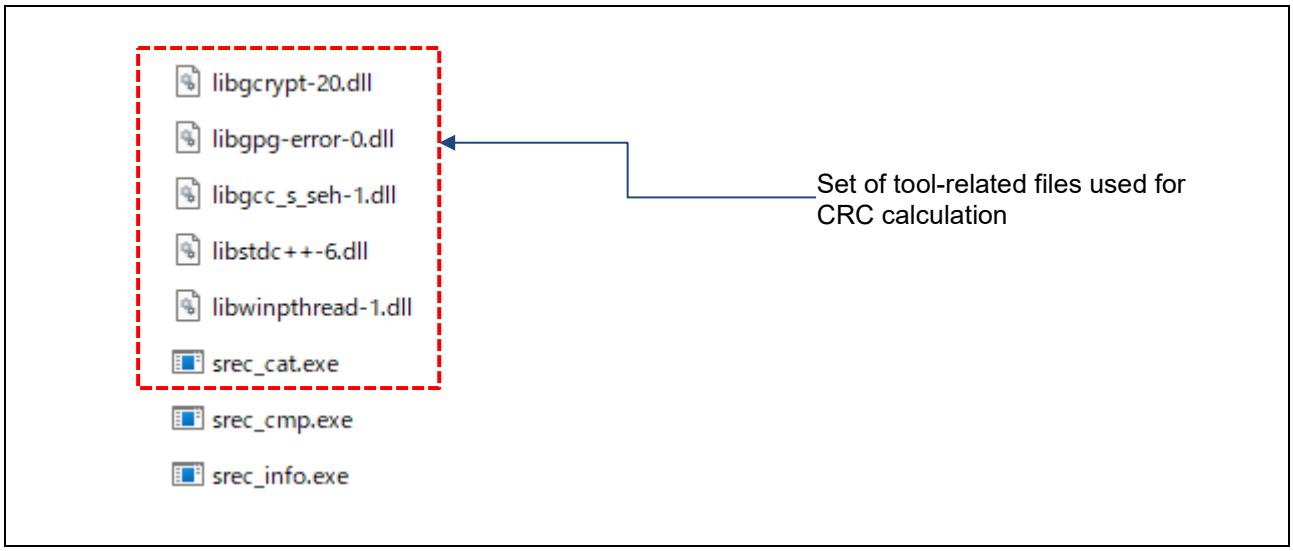

Figure 2.1 SRecord Tool Contents

An example of the folder structure of the project and SRecord tool is shown below.

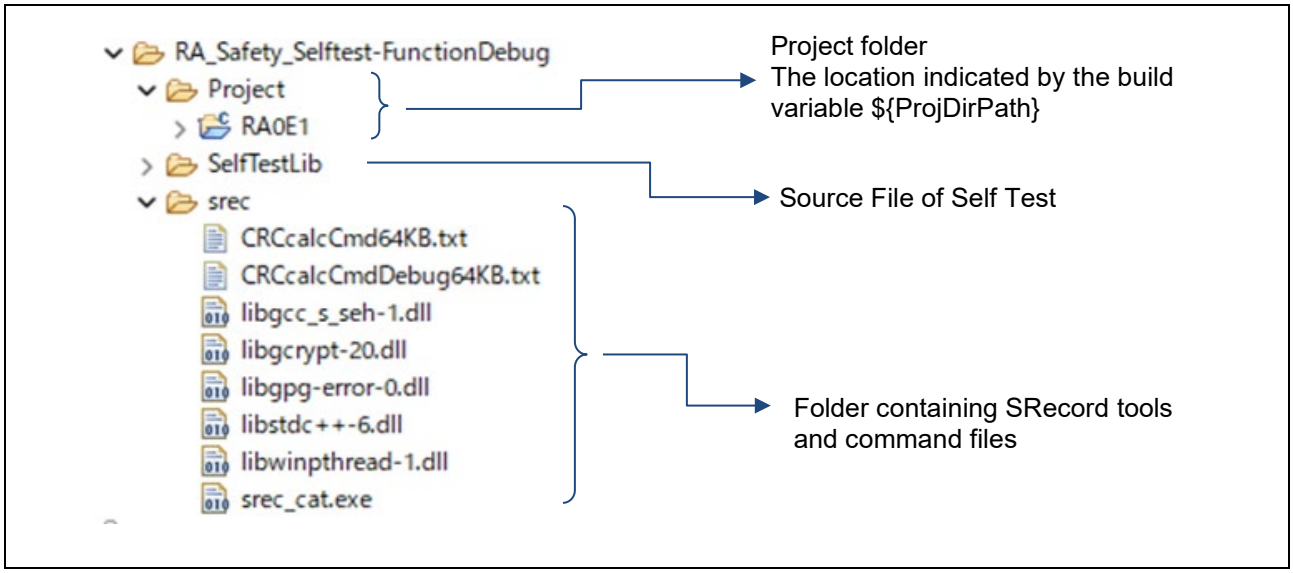

Figure 2.2 Folder Configuration Example

Open "project" ⇒ "property" of e^2 studio, and use the "objcopy" command in the step after building to generate an S record file from the .elf file. Here, the converted file name is Original.srec. This file is the input for the SRecord tool.

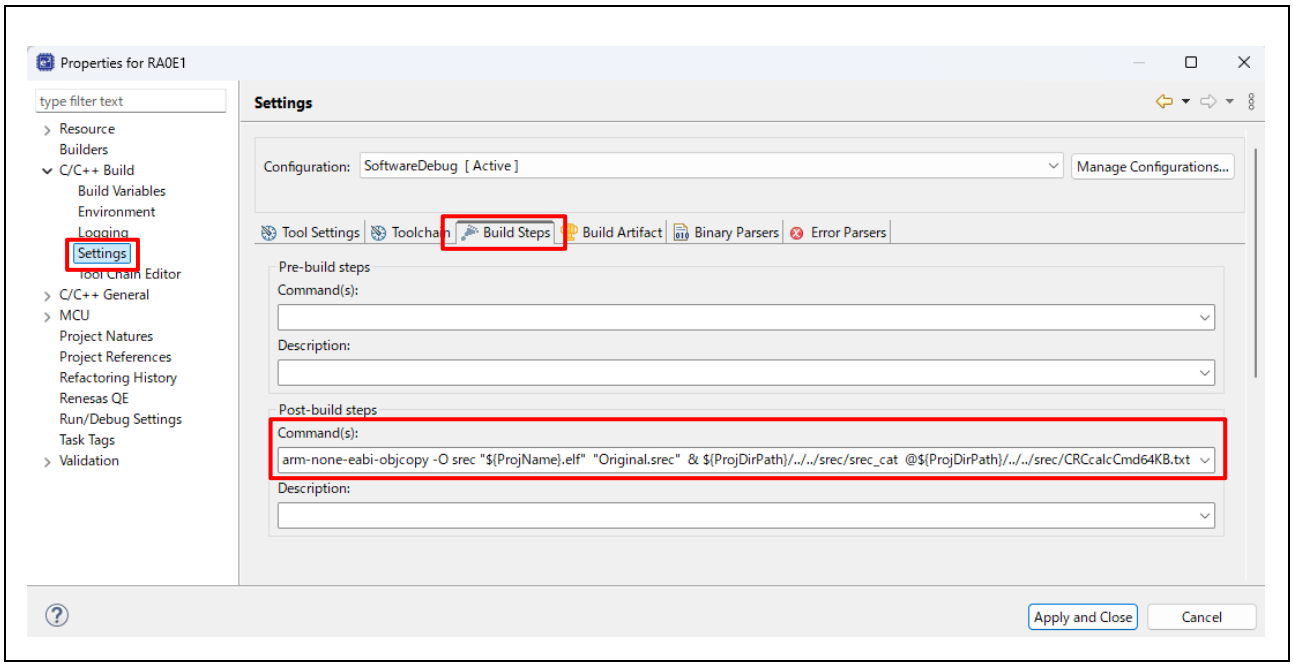

Figure 2.3 Output SRecord File and Start SRecord Tool

In the “Build Steps” tab of the ‘Settings’ screen in the figure above, describe the following in “Post-build steps”.

■ Example of Command(s): entry (write in one line without line breaks)

arm-none-eabi-objcopy -O srec "${ProjName}.elf" "**Original.srec**" & ${ProjDirPath}/../../srec/**srec_cat @**${ProjDirPath}/../../srec/**CRCcalcCmd64KB.txt**

The part before "&" on the first line indicates the generation of the S record file, and the description of "**srec_cat @**command file" indicates the start of the srec_cat tool.

An example of the description of "CRCcalcCmd64KB.txt" as a command file is shown below.

Please note that the CRC calculator equipped with RA0E1 only supports “LSB-first communication CRC generation”, the option specifications (red dotted frame section) in the following command file are different.

■ Contents of CRCcalcCmd64KB.txt file (example)

```
# CRC calculate
Original.srec                          # Read srec file
-fill 0xFF 0x00000 0x010000            # 64KB ROM fill by 0xFF
-crop 0x00000 0x00FFFC                 # CRC calculate area
# -STM32-le 0x00FFFC                   # Calculate and output CRC value
-CRC32_Little_Endian 0x00FFFC -CCITT   # CRC32 LSB (with initial value :
0xFFFFFFFF)
-crop 0x0FFFC 0x10000
Original.srec
-fill 0xFF 0x000000 0x0FFFC            # -fill 0xFF 0x0000 0x0FFFC
-Output_Block_Size 16                  # Set Block size
#-Output addcrc.srec
# Motorola S-Record fomat and the number of bytes which form each address is
"4".
-Output addcrc.srec -Motorola -address-length=4
```

If the ROM capacity varies depending on the device, change the address setting according to the device.

Besides, when debugging, some ROMs rewrite the contents of ROM due to a software break. In that case, it is necessary to set the operation target area to something other than the debug area.

With the above operation, **addcrc.srec** (S record file with CRC calculation result added to the end of program code) can be created in the build configuration folder under the project folder, so download it to the target board.

Open the debug configuration dialog box by selecting “Run” ⇒ “Debug Configurations” in e2 studio.

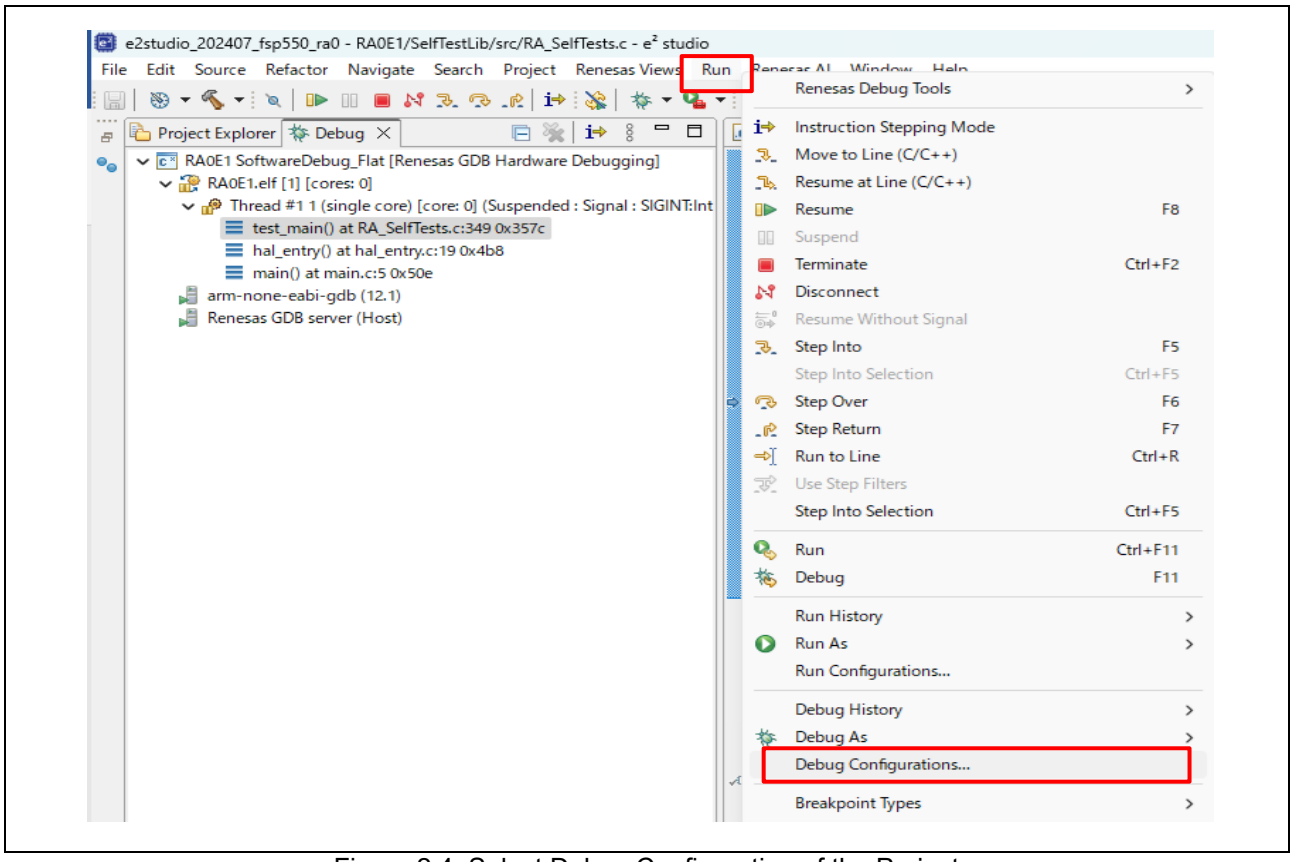

Figure 2.4 Select Debug Configuration of the Project

When the debug configuration dialog is displayed, select the "Startup" tab and select the build configuration to use. Only the symbol information is read from the ELF file, and the program image including the CRC calculation value is set to be read from **addcrc.srec**.

Click the "Debug" button to download the CRC calculation value to the target.

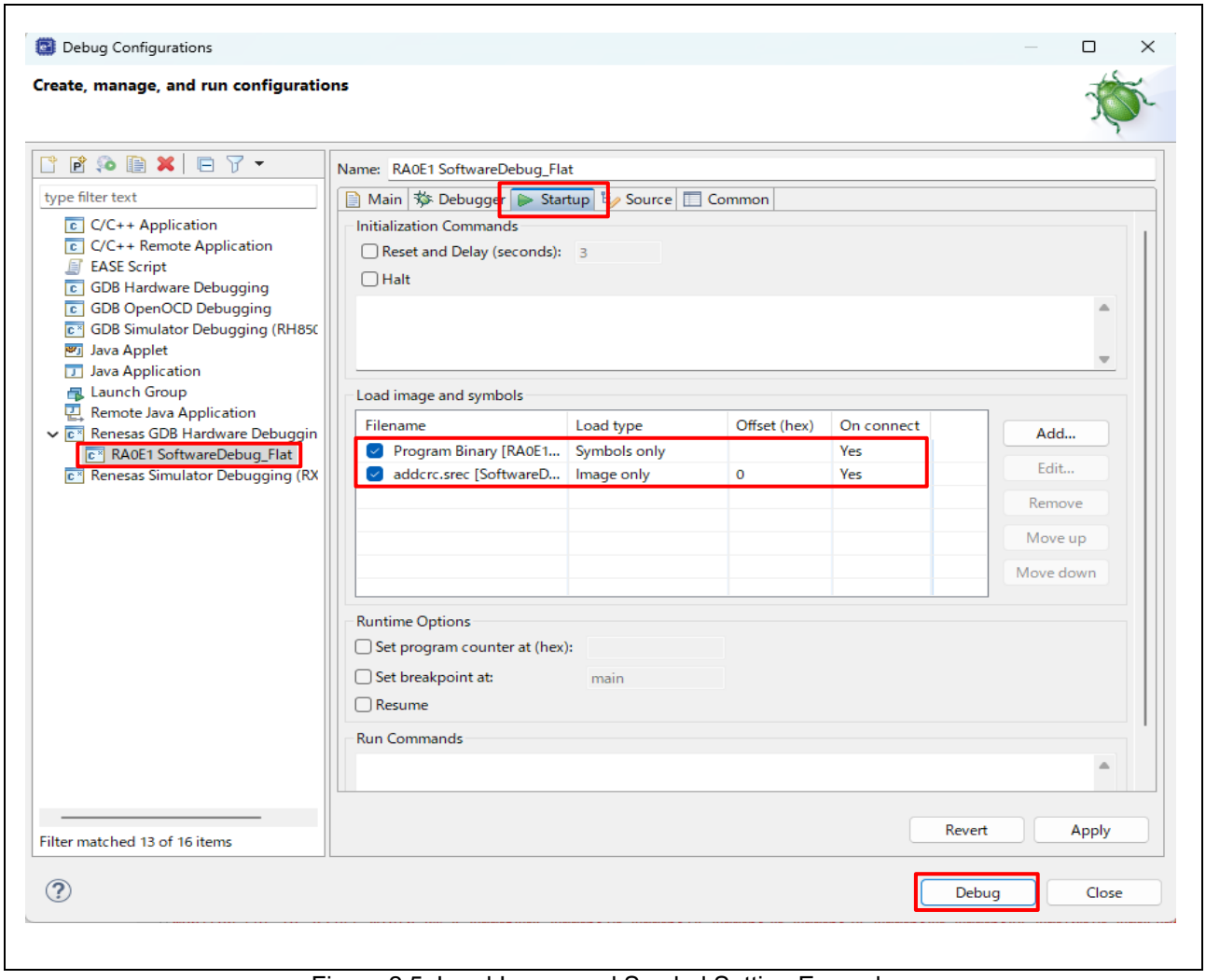

Figure 2.5 Load Image and Symbol Setting Example

2.2.2 Power-On

All the ROM memory used must be tested at power-on.

If this area is one contiguous block, function CRC_Calculate can be used to calculate and return a calculated CRC value.

If the ROM used is not in one contiguous block, the following procedure must be used.

1. Call CRC_Start.
2. Call CRC_AddRange for each area of memory to be included in the CRC calculation.
3. Call CRC_Result to get the calculated CRC value.

The calculated CRC value can then be compared with the reference CRC value stored in the ROM using function CRC_Verify.

It is user's responsibility to ensure that all ROM areas used by their project are included in the CRC calculations.

2.2.3 Periodic

It is suggested that the periodic testing of ROM is done using the CRC_AddRange, even if the ROM is contiguous. This allows the CRC value to be calculated in sections so that no single function call takes too long. Follow the procedure as specified for the power-on tests and ensure that each address range is small enough that a call to CRC_AddRange does not take too long.

2.3 RAM

It is very important to realize that the area of RAM that needs to be tested may change dramatically depending upon your project's memory map.

When testing RAM, keep the following points in mind:

1. RAM being tested cannot be used for anything else including the current stack.
2. Any non-destructive test requires a RAM buffer where memory contents can be safely copied to and restored from.
3. There are two stacks, Main and Process. Stack tests can test either or both of the two stacks. Any test of the stack requires a RAM buffer where the stack can be relocated to.

* The process stack area is not supported in the sample project because it has not been tested.

2.3.1 Power-On

At power-on, a full destructive test can be performed on the RAM other than the stack. The stack must be tested with a non-destructive test. However, if startup time is very important, it might be possible to fine tune this so that only the area of stack used before the power-on RAM test is performed using the slower non-destructive test and the rest of the stack tested with a destructive test.

2.3.2 Periodic

All periodic tests must be non-destructive. Because of periodic tests are called from interrupt handlers, it was tested assume that the device is in privileged mode.

2.4 Clock

To monitor the system clock, it is configuring settings related to TAU0 channel 5 in the FSP Configuration screen. It can check the settings related to the capture function of TAU0 channel 5 In the “Stacks Configuration” screen below.

The operation procedure is as follows:

・Double-click the **configration.xlm** file in the red frame in Project Explorer.

・The FSP Configuration screen will be displayed. Select the “Stacks” tab in the red frame to display the “**Stacks Configuration**” screen.

・Select “g_timer05 Timer, Independent Channel, ...” in Thread to select TAU0 ch5.

・Select the “**Properties”** tab.

The settings content(“Settings”) on the “**Stacks Configuration**” is displayed as the following figure.

Next, it selected each “**Common**”, “**Module g_timer05 Timer ...**”, “**General**” and “**Input**”, it is displayed the detailed setting contents.

The red dotted frame shows the main settings items of TAU0 ch5.

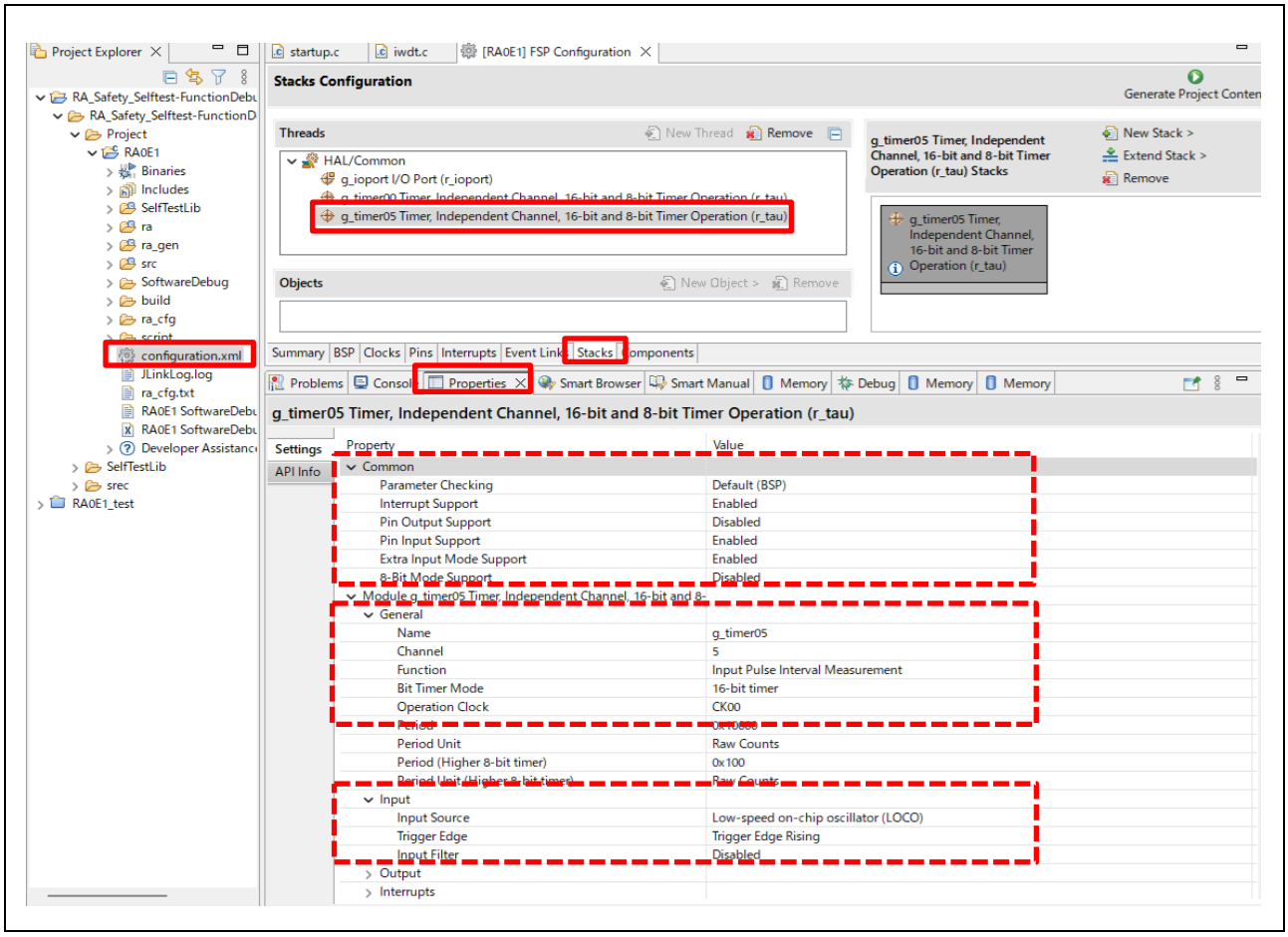

Figure 2.6 FSP Configuration Setting screen (Stacks Configuration tabs: TAU0 ch5)

For detailed specifications of TAU0 ch5, refer to the “Timer Array Unit (TAU)” chapter in the User's Manual: Hardware

Next, it can check the user name and event number of the function assigned to the capture interrupt of TAU0 channel 5 in the below figure “**Interrupts Configuration**” tab.

For detailed specifications on interrupts, refer to the “Interrupt Controller Unit (ICU)” chapter in the User's Manual: Hardware.

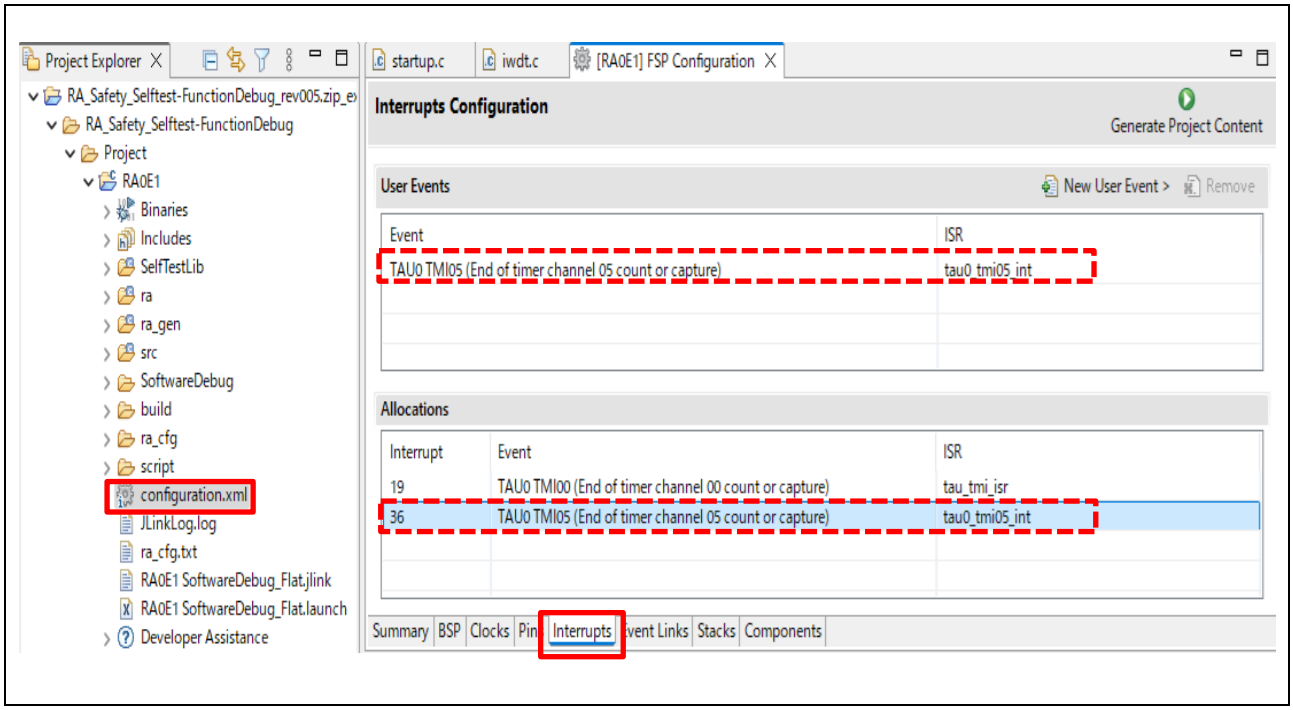

Figure 2.7 FSP Configuration Setting screen (Interrupts Configuration tabs : TAU0 ch5)

The above settings contents are reflected in the relevant registers when the “**ClockMonitor_Init**” function is called.

[Example]

```
    // TAU0 ch5 initial setting
    // In RA0, reflect the Stack setting in FSP Configuration to each TAU0 ch5 related register.
    // At the same time, set the error functions.
    ClockMonitor_Init ( Clock_Test_Failure );
```

In this sample project, the FSP configuration settings are reflected in the TAU0 ch5-related registers by this function.

In addition, the allowable range of clock frequencies is defined in clock_monitor.h as follows.

In this sample software, “LOCO” is set for the timer input of TAU0 ch5, and “PCLKB” (ICLK=HOCO selection) is set for the counter operation clock.

If it changed the above target clock, please change the following definitions to match the selected clock.

* The blue dotted frame below shows the upper and lower limits of the acceptable range in this sample software, and the green text shows supplementary comments.

[Definition parts in “clock_monitor.h”]

```
/* Hardware clock test reference values */
/*;hwMAXTIME= HOCO_FREQUENCY_MAX / LOCO_FREQUENCY_MIN */
/*;hwMINTIME= HOCO_FREQUENCY_MIN / LOCO_FREQUENCY_MAX
/*
;HOCO_FREQUENCY_MIN : Value considering -1% error of HOCO frequency (31.68MHz)
;LOCO_FREQUENCY_MAX: Value considering +15% error of low-speed on-chip
oscillator frequency (37.68KHz)
;HOCO_FREQUENCY_MAX: Value considering +1% error of HOCO frequency (32.32MHz)
;LOCO_FREQUENCY_MIN: Value considering -15% error of low-speed on-chip
oscillator frequency (27.85KHz)
;Change the lower and upper limits of the allowable range for pulse interval
measurement as appropriate depending on the system.
*/
// egde trig clock ; LOCO , count clock : PCLKB

#define hwMAXTIME (0x0489)        // 1161
#define hwMINTIME (0x0349)        //  841
```

2.5 Independent Watchdog Timer (IWDT)

In order to configure the Independent Watchdog Timer, it is necessary to set the OFS0 register in Option-Setting Memory. For example, suppose the Option-Setting Memory will set as follows.

Table 2.1 OFS0 Register Setting Example (IWDT Related)

Item	OFS0 Register Setting (For Example)
IWDT Start Mode Select (IWDTSTRT)	0: Automatically activate IWDT after a reset (auto start mode)
IWDT Timeout Period Select (IWDTTOPS[1:0])	11b : 2048 cycles
IWDT-Dedicated Clock Frequency Division Ratio Select (IWDTCKS[3:0])	1111b : 1/128
IWDT Window End Position Select (IWDTRPES[1:0])	11b : 0% (no window end position setting)
IWDT Window Start Position Select (IWDTRPSS[1:0])	11b : 100% (no window start position setting)
IWDT Reset Interrupt Request Select (IWDTRSTIRQS)	0: Enable non-maskable interrupt request or interrupt request
IWDT Stop Control (IWDTSTPCTL)	1: Stop counting when in Sleep, Snooze, or Software Standby mode.

When using FSP (Flexible Software Package) on e² studio, the "Option-Setting Memory" settings can be done in the property of the "BSP" tab of the configuration.

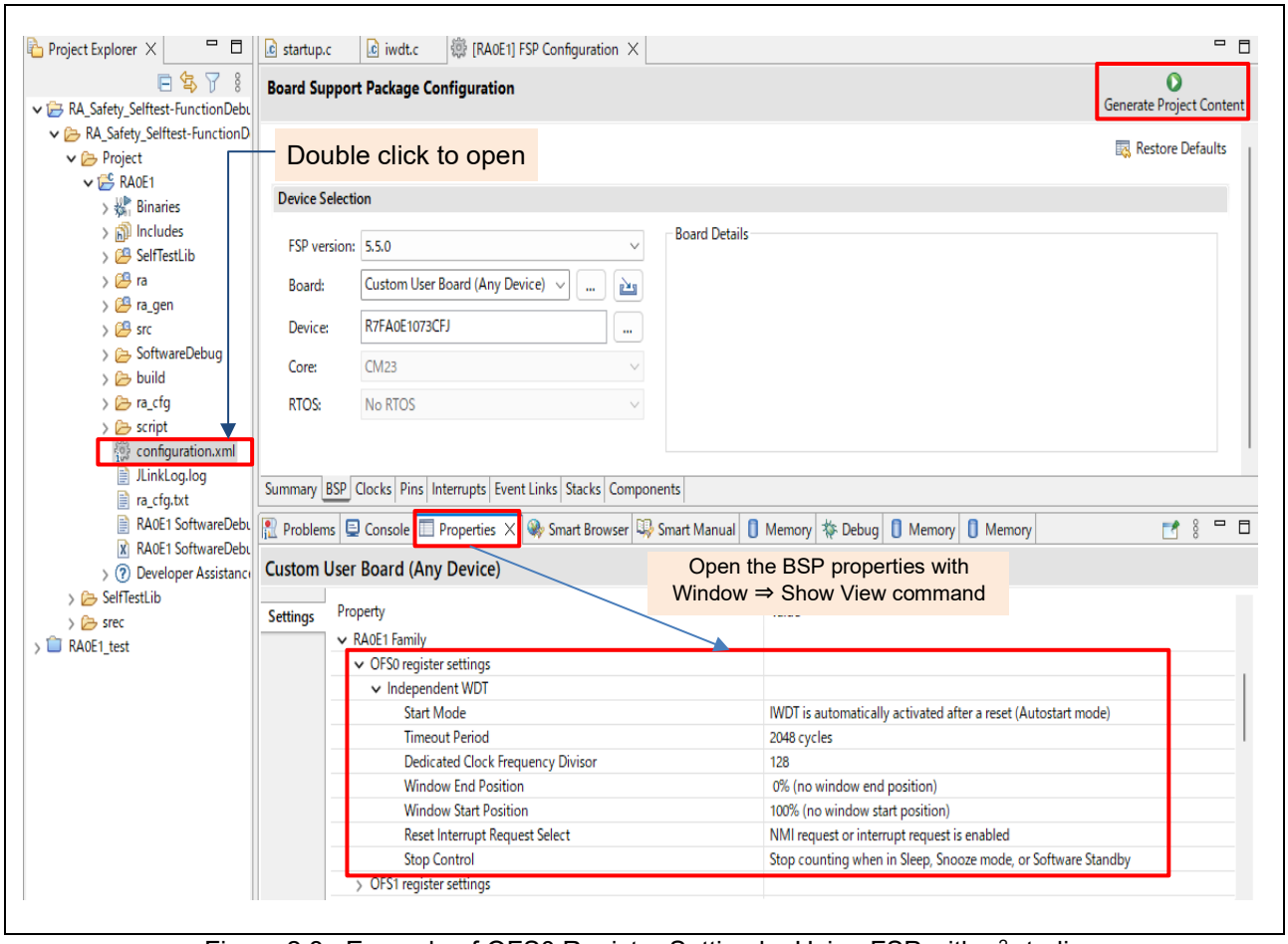

Figure 2.8. Example of OFS0 Register Setting by Using FSP with e² studio

When the "Generate Project Content" button is clicked, the contents set in the property will be reflected in the definition of the corresponding symbol in the following file. (For details, refer to "Renesas Flexible Software Package (FSP) User's Manual".)

- Applicable file

  ..*project-name*\ra_cfg\fsp_cfg\bsp**bsp_mcu_family_cfg.h**

- Applicable symbol (Excerpt)

```
#define OFS_SEQ1 0xA001A001 | (0 << 1) | (3 << 2)
#define OFS_SEQ2 (15 << 4) | (3 << 8) | (3 << 10)
#define OFS_SEQ3 (0 << 12) | (1 << 14) | (1 << 17)
    :      :
```

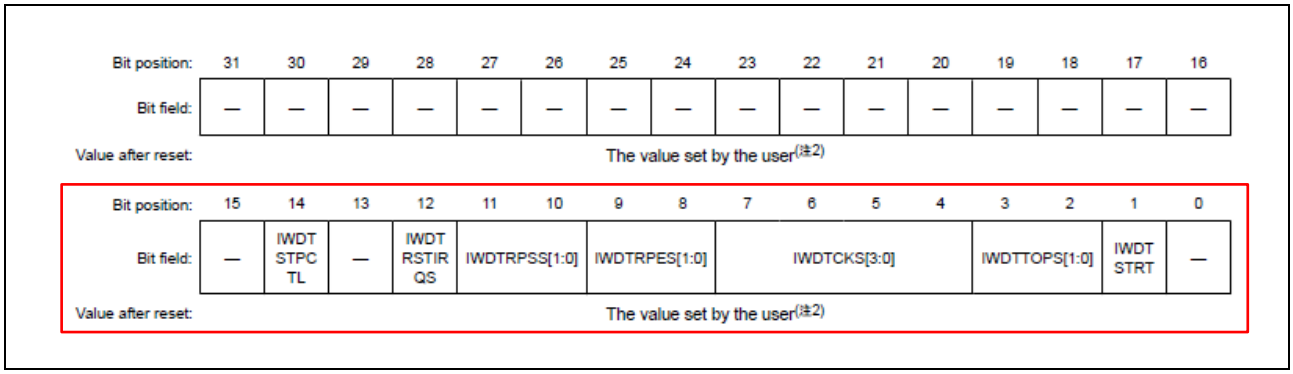

Figure 2.9 Option Function Select Register 0 (OFS0)

For detailed specifications on IWDT, refer to the “Independent Watchdog Timer (IWDT)” chapter in the User's Manual: Hardware.

The Independent Watchdog Timer should be initialized as soon as possible following a reset with a call to IWDT_Init:

```
/*Setup the Independent WDT.*/
IWDT_Init();
```

After this, the watchdog timer must be refreshed regularly enough to prevent the watchdog timer timing out and performing a reset. Note, if using windowing the refresh must not just be regular enough but also timed to match the specified window. A watchdog timer refresh is called by calling this:

```
/*Regularly kick the watchdog to prevent it performing a reset. */
IWDT_Kick();
```

If the watchdog timer has been configured to generate an NMI on error detection then the user must handle the resulting interrupt.

2.6 ADC

This ADC (ADC12) has a function that allows you to select the + side reference voltage, - side reference voltage, or internal reference voltage for test A/D conversion.

Use the following definition symbols to test the ADC module whether falls within the defined tolerance range(blue background color).

* Users should modify each definition according to their system.

In a calibrated system, this tolerance can be made stricter.

[Definition parts in “test_adc.h”]

```
// See "31.6.1 "A/D conversion characteristics" on UMH of RA0E1
#define OVERALL_ERROR_LSB_UNIT (9)                          /* AD Converter Overall error */
#define VSS_RANGE_MAX          (0 + OVERALL_ERROR_LSB_UNIT)  /* Vss Range Max */
#define VSS_RANGE_MIN          (0x0000)                     /* Vss Range Min */
#define AD_RESOLUTION_HEX      (0x0FFF)                     /* AD Converter Resolution */
#define VDD_RANGE_MIN          (AD_RESOLUTION_HEX - OVERALL_ERROR_LSB_UNIT)  /* Vdd Range Min */
#define VDD_RANGE_MAX          (AD_RESOLUTION_HEX)          /* Vdd Range Max */

/* Please change it according to the VDD voltage. */
#define VDD           (3.3)      /* Vdd */
//#define VDD           (5.0)      /* Vdd */
#define VBGR_MIN      (1.4)      /* Vbgr Min */
#define VBGR_MAX      (1.56)     /* Vbgr Max */
#define VBGR_RANGE_MIN ((((VBGR_MIN * AD_RESOLUTION_HEX)/VDD)) - OVERALL_ERROR_LSB_UNIT) /* Vdd
Range Min */
#define VBGR_RANGE_MAX ((((VBGR_MAX * AD_RESOLUTION_HEX)/VDD)) + OVERALL_ERROR_LSB_UNIT) /* Vdd
Range Max */
```

For details on the above errors and internal reference voltage values, refer to the User's Manual: Hardware, “Analog Characteristics.”

When performing this ADC test, it is necessary to call Test_ADC_Init to initialize.

2.6.1 Power-On

At power-on, the ADC module can be tested using the Test_ADC_Wait function.

In this function, either the “-“side reference voltage, or the “+” side reference voltage or internal reference voltage for test A/D conversion is performed, and the function waits until the AD conversion is completed.

The return value of this function must be checked for the result.

2.6.2 Periodic

Periodical tests perform AD conversion by calling T**est_ADC_Start**.

When this function is called, the ADC module switches to the test A/D conversion target.

* This ADC module does not have automatic rotation.

Therefore, after calling **Test_ADC_Start**, it is necessary to check the AD conversion results to call **Test_ADC_CheckResult**.

2.7 GPIO

The specification of the port to be test target in each device is defined by the **g_GPIO_Test_Port** structure in the **gpio_config.h** header file (port m, bit n).

```
// Port specification example
static const struct {
   uint16_t m;
   uint16_t n;
} g_GPIO_Test_Port[GPIO_TEST_NUM] =
{
   { 0 ,  12 } ,     // PORT012
   { 1 ,  0 } ,      // PORT100
   { 2 ,  1 }        // PORT201
};
```

Website and Support

Visit the following URLs to learn about the key elements of the RA MCU, download tools, components, and related documentation, and get support.

- RA Product Information: www.renesas.com/ra
- RA FSP (Flexible Software Package): www.renesas.com/FSP
- RA Support Forum: www.renesas.com/ra/forum
- Renesas Support: www.renesas.com/support

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Aug 25, 2025	–	First edition

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.