RL78 Family

IEC60730/60335 Self Test Library for RL78/L23 MCU CC-RL

R01AN8017EJ0100
Rev.1.00
30.Sep.2025

Introduction

Today, as automated electronic control 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.

This application note provides guidelines on how to assist in compliance with the IEC60730 Class B safety standard through the use of flexible sample software routines. 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 during program execution. This document and the accompanying sample code provide examples of how to do this.

Target Devices

RL78/L23 microcontroller

Contents

1. Self Test Libraries Introduction

The Self-Test Library (STL) consists of self-test functions for the CPU registers, internal memory, and the system clock. As described below, the test harness provides an application program interface (API) for each module that performs a self-test. Use each function according to its purpose.

The self-test library functions are divided into modules according to VDE certification. The CS+ test harness allows each of the test functions to be selected in turn and run as a stand-alone function.

A hardware requirement of the system is the availability of two or more independent clock sources (such as a crystal/ceramic oscillator and an independent oscillator or an external input source). This is necessary to set up another clock reference to monitor the system clock. The RL78 meets this requirement by using fast and slow internal oscillators that operate independently of each other.

The application can provide a more accurate external reference clock or use it as the main system clock. It is also possible to use an external crystal/resonator.

The following CPU self test functions are included in the RL78 self test library.

- CPU Registers
  The following CPU registers are tested:
  All CPU general-purpose registers in all four register banks, Stack Pointer (SP), Program Status Word (PSW), Extended Registers (ES and CS), and Program Counter (PC).
  The internal data paths are verified as part of the correct operation test of the above registers.
  IEC Reference-IEC 60730-1:2013+A1:2015+A2:2020 Annex H - Table H2.16.5.

- Invariable Memory
  This tests the MCU internal Flash memory.
  IEC Reference-IEC 60730-1:2013+A1:2015+A2:2020 Annex H – H2.19.4.1 CRC – Single Word.

- Variable Memory
  This tests the Internal SRAM memory.
  IEC Reference-IEC 60730-1:2013+A1:2015+A2:2020 Annex H – H2.19.6.2 marching memory test.

- System Clock
  Verifies the system clock operation and correct frequency against a reference clock source (Note this test requires the use of an internal or external independent reference clock)
  IEC Reference-IEC 60730-1:2013+A1:2015+A2:2020 Annex H – H2.18.10.1 frequency monitoring.

- CPU/Program Counter
  To check that the program is executing the sequence within the specified time, a built-in watchdog timer that runs on a clock independent of the CPU is used. The test harness is equipped with a process to monitor whether the sequence is being executed in the expected order.
  IEC Reference-IEC 60730-1:2013+A1:2015+A2:2020 Annex H – H2.18.10.3 time-slot and logical monitoring.

2. Self Test Library Functions

2.1 CPU Register Tests

This section describes CPU register tests routines. The test harness control file ‘main.c’ provides examples of the API for each of the CPU register tests using “C” language.

These modules test the fundamental aspects of the CPU operation. Each of the API functions has a return value in order to indicate the result of a test.

Each of the test modules saves the original contents of the register(s) under test and restores the contents on completion.

The target CPU registers are as follows:

- General-Purpose Register: AX, HL, DE, BC of Register Banks 0 - 3

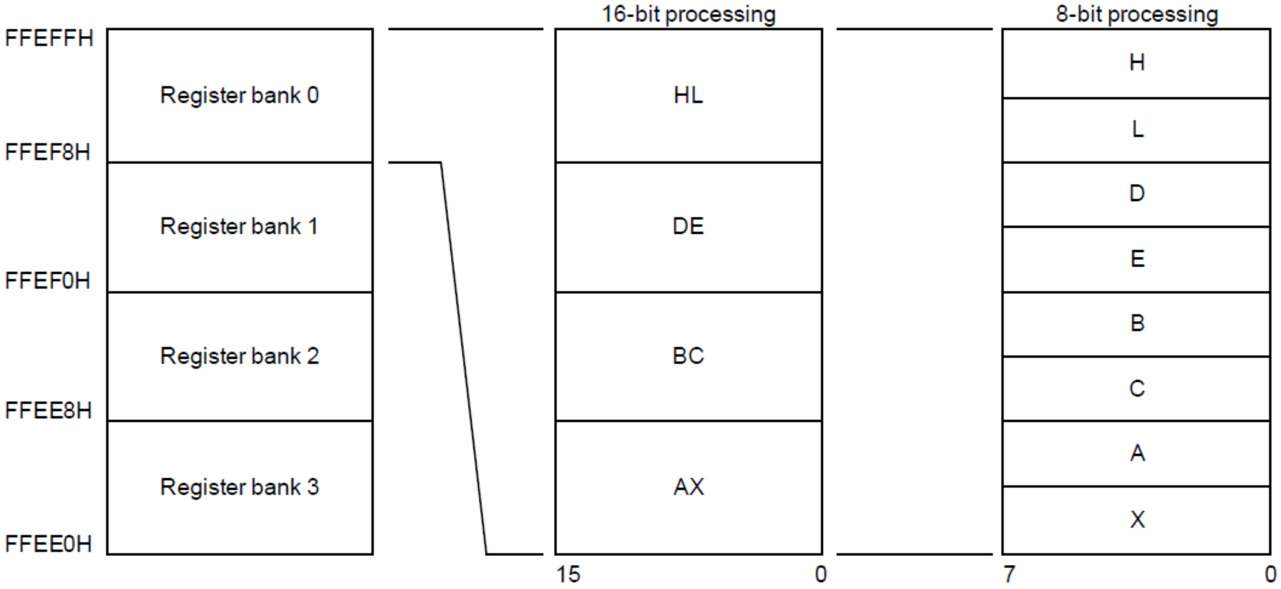


Figure 2-1 General-Purpose Registers Configuration

- Stack Pointer (SP)

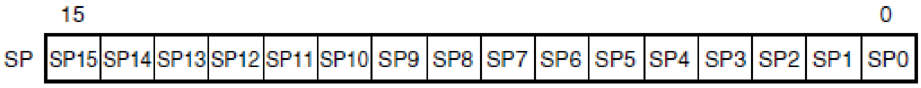


Figure 2-2 Stack Pointer Configuration

- Program Status Word (PSW)

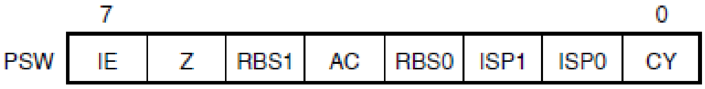


Figure 2-3 PSW Register Configuration

- Code Address Extension Register (CS)

	7	6	5	4	3	2	1	0
CS	0	0	0	0	CS3	CP2	CP1	CP0

Figure 2-4 CS Register Configuration

- Data Address Extension Register (ES)

	7	6	5	4	3	2	1	0
ES	0	0	0	0	ES3	ES2	ES1	ES0

Figure 2-5 ES Register Configuration

- Program Counter (PC)

	19	0
PC		

Figure 2-6 Program Counter Configuration

2.1.1 CPU Register Tests - Software API

Table 2-1 Source files: CPU Working Registers Tests

STL File name	Header Files
stl_RL78_RegisterTest.asm	stl.h
Test Harness File Names	Header Files
main.c	

Syntax	
char stl_RL78_RegisterTest(void)	
Description	
This module tests the RL78 general-purpose registers. Registers AX, HL, DE, BC in all three register banks (Banks 0, 1, 2, 3) These registers are tested as16bit registers. 1. Write h'5555 to the register being tested. 2. Read the register and verify that it is equal to the value written. 3. Write h'AAAA to the register being tested. 4. Read the register and verify that it is equal to the value written. The calling function must not allow interrupts to occur during this test and must begin this test with register bank 0 selected. The register contents before the test are restored after the test is completed. The function RegisterTest_Failure is called by the test harness control file (main.c) when an error is detected.	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
char	Test result 0 = Test passed. 1 = The test or parameter check failed.

Table 2-2 Source files: CPU Registers Tests – PSW

STL File name	Header Files
stl_RL78_RegisterTest_psw.asm	stl.h
Test Harness File Names	Header Files
main.c	

Syntax	
char stl_RL78_RegisterTest_psw(void)	
Description	
Test the 8bit Program Status Word (PSW) register. The following tests are performed: 1. Write h'55 to the register being tested. 2. Read the register and verify that it is equal to the value written. 3. Write h'AA to the register being tested. 4. Read the register and verify that it is equal to the value written. The calling function must not allow interrupts to occur during this test. The register contents before the test are restored after the test is completed. The function RegisterTest_Failure is called by the test harness control file (main.c) when an error is detected.	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
char	Test result 0 = Test passed. 1 = The test or parameter check failed.

Table 2-3 Source files: CPU Registers Tests - SP

STL File name	Header Files
stl_RL78_RegisterTest_stack.asm	stl.h
Test Harness File Names	Header Files
main.c	

Syntax	
char stl_RL78_RegisterTest_stack(void)	
Syntax	
Test the 16bit Stack Pointer (SP) register. The following tests are performed: 1. Write h'5555 to the register being tested. 2. Read the register and verify that it is equal to h'5554. 3. Write h'AAAA to the register being tested. 4. Read the register and verify that it is equal to the value written. The calling function must not allow interrupts to occur during this test. The register contents before the test are restored after the test is completed. The function RegisterTest_Failure is called by the test harness control file (main.c) when an error is detected.	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
char	Test result 0 = Test passed. 1 = The test or parameter check failed.

Table 2-4 Source files: CPU Registers Tests - CS

STL File name	Header Files
stl_RL78_RegisterTest_cs.asm	stl.h
Test Harness File Names	Header Files
main.c	

Syntax	
char stl_RL78_RegisterTest_cs(void)	
Description	
Test the 8bit code extension (CS) register The following tests are performed: 1. Write h'05 to the register being tested. 2. Read the register and verify that it is equal to the value written. 3. Write h'0A to the register being tested. 4. Read the register and verify that it is equal to the value written. Please note that the top 4 bit are fixed to “0” The calling function must not allow interrupts to occur during this test. The register contents before the test are restored after the test is completed. The function RegisterTest_Failure is called by the test harness control file (main.c) when an error is detected.	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
char	Test result 0 = Test passed. 1 = The test or parameter check failed.

Table 2-5 Source files: CPU Registers Tests - ES

STL File name	Header Files
stl_RL78_RegisterTest_es.asm	stl.h
Test Harness File Names	Header Files
main.c	

Syntax	
char stl_RL78_RegisterTest_es(void)	
Description	
Test the 8bit data extension (ES) register. The following tests are performed: 1. Write h'05 to the register being tested. 2. Read the register and verify that it is equal to the value written. 3. Write h'0A to the register being tested. 4. Read the register and verify that it is equal to the value written. Please note that the top 4 bit are fixed to “0”. The calling function must not allow interrupts to occur during this test. The register contents before the test are restored after the test is completed. The function RegisterTest_Failure is called by the test harness control file (main.c) when an error is detected.	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
char	Test result 0 = Test passed. 1 = The test or parameter check failed.

Table 2-6 Source files: CPU Registers Tests - PC

STL File name	Header Files
stl_RL78_RegisterTest_pc.asm	stl.h
Test Harness File Names	Header Files
main.c	

Syntax	
char stl_RL78_RegisterTest_pc(void)	
Description	
Test the program counter (PC) register. The following tests are performed: 1. Call the program counter (PC) test function using call instruction 2. The test function sets the return value to the result of reversing the arguments and returns 3. After calling the test function with the call instruction, verify that the return value matches the expected value. The calling function must not allow interrupts to occur during this test. The register contents before the test are restored after the test is completed. The function RegisterTest_Failure is called by the test harness control file (main.c) when an error is detected.	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
char	Test result 0 = Test passed. 1 = The test or parameter check failed.

2.2 Invariable Memory Test – Flash ROM

This section describes the Flash memory test using CRC routines. 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 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 popular because they are simple to implement in binary hardware and are easy to analyse mathematically.

The Flash ROM test can be verified by generating a reference CRC value for the contents of the ROM and storing this in memory. During the memory self test the same CRC algorithm is used to generate a CRC value, which is compared with the reference CRC value. The technique recognises all one-bit errors and a high percentage of multi-bit errors.

The complication with using CRC is that the CRC value must be pre-generated and compared with other CRC values generated by different CRC generators. This is not an easy task since there are many factors that affect the resulting CRC value even if the basic CRC algorithm is the same. In practice, the order in which the data is presented to the algorithm, the expected bit order in any lookup table used, and the required bit order in the actual CRC value are all interrelated. The hardware self-test functions can be repeated, allowing the full CRC value to be calculated or partial CRC values to be calculated depending on the desired application behaviour. The full CRC calculation (or the first of multiple calculations) uses h'0000 as the initial value. Multiple calculations use the previous result as the initial value for the next calculation.

The hardware module is a generic CRC function built into the RL78. The hardware module essentially uses the same CRC algorithm to calculate the CRC value with a different data format (LSB first).

2.2.1 CRC16-CCITT Algorithm

The RL78 CRC module supports CRC16-CCITT. When the CRC module is driven by this software, the following 16-bit CRC16-CCITT is generated.

Hardware Algorithm

- CCITT 16 Polynomial = 0x1021 ($x^{16} + x^{12} + x^5 + 1$)
- LSB first (The bit order is inverted when input and the operation is performed, and the bit order of the operation result is also inverted when outputting)
- Input data width = 8 Bits
- Initial value = 0x0000 or the 16-bit result of the previous partial CRC calculation

2.2.2 Hardware CRC - Software API

Table 2-7 Source files: Hardware CRC Calculation

STL FILE NAME	Header Files
stl_RL78_peripheral_crc.asm	stl.h
Test Harness File Names	Header Files
main.c	

Syntax	
unsigned short stl_RL78_peripheral crc(unsigned short gcrc, CHECKSUM_CRC_TEST_AREA *p)	
Description	
Uses the hardware CRC peripheral (general-purpose CRC) to calculate the CRC value for the specified address range. The starting address and calculation range (length) are passed from the calling function in the structure shown in the table below. The returned results are partial or full values depending on the specified parameters. The test harness control file (main.c) calculates the CRC of the entire inspection area and then checks the CRC value obtained at build time. If they do not match, the function “ROM_Test_Failure” is called to process the test result.	
Input Parameters	
unsigned short gcrc	Initial value for CRC calculation
CHECKSUM_CRC_TEST_AREA *p	A pointer to a structure that stores the start address and calculation range.
Output Parameters	
NONE	N/A
Return Values	
unsigned short	16-bit calculated CRC value (Full or partial result) CPU register AX

Source files: Hardware CRC Parameter Structure

Syntax	
static CHECKSUM_CRC_TEST_AREA checksum_crc;	
Description	
A structure declaration and instance providing parameters passed to the hardware CRC module (stl_RL78_peripheral_crc.asm) from the calling function in main.c	
Input Parameters	
Unsigned long m_length	The memory range to test (length = number of bytes)
unsigned long m_start_address	Start address of CRC calculation
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

2.3 Variable memory – SRAM

The March test is a test regime that is widely recognized as an effective method for testing RAMs.

A March test consists of a finite set of March elements, which in turn consist of a finite set of operations performed on each cell of the memory array. In general, adding more elements to an algorithm increases its fault coverage but slows down its execution time.

The algorithm itself is destructive and does not save the current RAM values. Therefore, if you want to test after the application system is initialized or running, you must save the RAM contents. The March C- and March X test modules test only a portion of the RAM, so you do not need to have a large temporary area to store the data during testing. The added test modules ("stl_RL78_march_c_initial" and "stl_RL78_march_x_initiarl") before the system is initialized, allowing you to test the entire memory area before starting the main application.

The RAM area being tested cannot be used for any other purpose during the test, which makes testing RAM used as a stack particularly difficult: this area can only be tested before the application’s C stack is initialized or after the application has finished processing.

The following chapters describe each of the March tests.

2.3.1 Algorithms

(1) March C-

March C- algorithm (van de Goor 1991) is a set of six March elements that perform a total of 10 operations.

Stuck-at Fault (SAF)

- The logic value of a cell or a line is always 0 or 1.

Transition Fault (TF)

- A cell or a line that fails to undergo a 0→1 or a 1→0 transition.

Coupling Fault (CF)

- A write operation to one cell changes the content of a second cell.

Address Decoder Fault (AF)

- Any fault that affects address decoding.
- With a certain address, no cells can 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.

The usual March C- algorithm employs 6 March elements.

1. ⇔(w0) : Write all zeros to array
2. ↑(r0,w1) : Starting at lowest address, read zeros, write ones, increment up array bit by bit.
3. ↑(r1,w0) : Starting at lowest address, read ones, write zeros increment up array bit by bit.
4. ↓(r0,w1) : Starting at highest address, read zeros, write ones, decrement down array bit by bit.
5. ↓(r1,w0) : Starting at highest address, read ones, write zeros, decrement down array bit by bit.
6. ⇔(r0) : Read all zeros from array.

Note: The symbols in the march element represent the following processes:

⇔ : Perform the operation in ascending or descending order of addresses.

↑ : Perform the operation in ascending address order.

↓ : Perform the operation in descending order of addresses.

w0: write 0 to cell.

w1: write 1 to cell.

r0: Read the cell and check if the value is 0.

r1: Read the cell and check if the value is 1.

(2) March X

The March X algorithm has a simple structure and is fast, but it is not suitable for detailed testing because it has only four March elements and four types of overall processing.

- Stuck-at Faults (SAF)
- Transition Faults (TF)
- Reverse Coupling Faults (Cfin)
- Address Decoder Faults (AF)

It uses four march elements:

1. ⇔(w0) : Write all zeros to array.
2. ↑(r0,w1): Starting at lowest address, read zeros, write ones, increment up array bit by bit.
3. ↓(r1,w0): Starting at highest address, read ones, write zeros, decrement down array bit by bit.
4. ⇔(r0) : Read all zeros from array.

2.3.2 Variable Memory Test – Software API

2.3.2.1 System March C-

The system March C- test is designed to run after the application system has been initialised and is executed using normal function call from the test harness, thus using some C stack resources. The module can be used to test part or all of the RAM area, but as the test is destructive, care should be taken to buffer the area being tested Therefore it is not advised to use this module to test the whole RAM memory area in a single operation. In addition, make sure not to destroy the RAM area used by this test itself as the stack area.

This test is configured to use 8-bit RAM accesses and can allow perform byte-by-byte testing. However, for all fault types to be detected it is important to test a data range bigger than tow bytes.

Table 2-8 Source files: System March C-

STL FILE NAME	Header Files
stl_RL78_march_c.asm	stl.h
Test Harness File Names	Header Files
main.c	

Syntax	
char stl_RL78_march_c (unsigned char *addr, unsigned short num)	
Description	
Use the March C- algorithm to test the RAM address range specified by the calling function and returns the result (pass/fail). This module runs after the application system is initialized. The function RAM_Test_Failure is called by the test harness control file (main.c) when an error is detected.	
Input Parameters	
unsigned char *addr	Pointer to the start address of the RAM to be tested.
unsigned short num	The range (Number of bytes) of the RAM to be tested.
Output Parameters	
NONE	N/A
Return Values	
char	Test result 0 = Test passed. 1 = The test or parameter check failed.

2.3.2.2 System March X

The system March X self test function is the essentially the same as the system March C- module except that it only implements the reduced March X algorithm. The module is designed to run after the application system has been initialised and so should not be used to test the whole memory area in a single operation. In addition, make sure not to destroy the RAM area used by this test itself as the stack area.

This test is configured to use 8-bit RAM accesses and can allow perform byte-by-byte testing. However, for all fault types to be detected it is important to test a data range bigger than tow bytes.

Table 2-9 Source files: System March X

STL FILE NAME	Header Files
stl_RL78_march_x.asm	stl.h
Test Harness File Names	Header Files
main.c	

Syntax	
char stl_RL78_march_x (unsigned char *addr, unsigned short num)	
Description	
Use the March X algorithm to test the RAM address range specified by the calling function and returns the result (pass/fail). This module runs after the application system is initialized. The function RAM_Test_Failure is called by the test harness control file (main.c) when an error is detected.	
Input Parameters	
unsigned char *addr	Pointer to the start address of the RAM to be tested.
unsigned short num	The range (Number of bytes) of the RAM to be tested.
Output Parameters	
NONE	N/A
Return Values	
char	Test result 0 = Test passed. 1 = The test or parameter check failed.

2.3.2.3 Initial March C-

The initial March C- test is designed to run before the application system has been initialized and is executed without using function calls from the test harness. The test is launched via a “jump” from the modified “cstart.asm” module and return to “cstart.asm” module is also performed via a “jump”. The test status result is contained in the 8bit accumulator (A). This module is designed to provide a complete RAM test before the system is started up, and the “C” environment is initialized.

This test function is configured to use 8-bit RAM access.

Table 2-10 Source File: Initial March C-

STL FILE NAME	Header Files
stl_RL78_march_c_initial.asm	NONE
Test Harness File Names	Header Files
cstart.asm	NONE

Syntax	
stl_RL78_march_c_initial	
Description	
Uses the March C-algorithm to test the RAM address range specified by the calling function and this module runs before the application system is initialized and returns the result (pass/fail). No function calls are used. The function “stl_RL78_InitialRamTestResult” is called to notify the test result. Note: The function “stl_RL78_InitialRamTestResult” has an example in the test harness control file (main.c).	
Input Parameters	
CPU Register AX	16bit Register holding the start address of the RAM to be tested.
CPU Register BC	16bit Register holding the range (Number of bytes) of the RAM to be tested.
Output Parameters	
NONE	N/A
Return Values	
CPU Register A	Test result 0 = Test passed. 1 = The test or parameter check failed.

2.3.2.4 Initial March X

The initial March X test is designed to run before the application system has been initialized and is executed without using function calls from the test harness. The test is launched via a “jump” from the modified “cstart.asm” module and return to “cstart.asm” module is also performed via a “jump”. The test status result is contained in the 8bit accumulator (A). This module is designed to provide a complete RAM test before the system is started up, and the “C” environment is initialized

This test function is configured to use 8bit RAM accesses.

Table 2-11 Source File: Initial March X

STL FILE NAME	Header Files
stl_RL78_march_x_initial.asm	NONE
Test Harness File Names	Header Files
cstart.asm	NONE

Syntax	
stl_RL78_march_x_initial	
Description	
Uses the March X algorithm to test the RAM address range specified by the calling function and this module runs before the application system is initialized and returns the result (pass/fail). No function calls are used. The function “stl_RL78_InitialRamTestResult” is called to notify the test result. Note: The function “stl_RL78_InitialRamTestResult” has an example in the test harness control file (main.c).	
Input Parameters	
CPU Register AX	16bit Register holding the start address of the RAM to be tested.
CPU Register BC	16bit Register holding the range (Number of bytes) of the RAM to be tested.
Output Parameters	
NONE	N/A
Return Values	
CPU Register A	Test result 0 = Test passed. 1 = The test or parameter check failed.

2.3.2.5 Stack Area Test (March C-)

Uses C stack resources to run normal function calls from the test harness. It is possible to test the entire STACK area. Since the test is destructive, the current state is saved to a buffer before testing. By changing the offset of the STACK_TEST_AREA parameter for each test, it is possible to test only a part of it.

RAM testing is performed using System March C-.

Table 2-12 Stack Area Test (March C-)

STL FILE NAME	Header Files
stl_RL78_RamTest_Stacks_c.asm	stl.h
Test Harness File Names	Header Files
main.c	

Syntax	
char stl_RL78_RamTest_Stacks_c (STACK_TEST_AREA *p)	
Description	
It switches the stack pointer (SP) to the specified area, tests the address range of the specified buffer RAM using the March C-algorithm, and if the result (pass/fail) is pass, copies the contents of the stack area to the buffer RAM. It then tests the stack area using the March C-algorithm, restores the contents saved in the buffer RAM and the stack pointer (SP). It then returns the test result (pass/fail). This module runs after the application system is initialized. The function RAM_Test_Failure is called by the test harness control file (main.c) when an error is detected.	
Input Parameters	
STACK_TEST_AREA *p	A pointer to a structure that stores the buffer RAM, its size, and the new stack area.
Output Parameters	
NONE	N/A
Return Values	
char	Test result 0 = Test passed. 1 = The test or parameter check failed.

2.3.2.6 Stack Area Test (March X)

Uses C stack resources to run normal function calls from the test harness.

It is possible to test the entire STACK area. Since the test is destructive, the current state is saved to a buffer before testing.

By changing the offset of the STACK_TEST_AREA parameter for each test, it is possible to test only a part of it.

RAM testing is performed using System March X.

Table 2-13 Stack Area Test (March X)

STL FILE NAME	Header Files
stl_RL78_RamTest_Stacks_x.asm	stl.h
Test Harness File Names	Header Files
main.c	

Syntax	
char stl_RL78_RamTest_Stacks_x (STACK_TEST_AREA *p)	
Description	
It switches the stack pointer (SP) to the specified area, tests the address range of the specified buffer RAM using the March X algorithm, and if the result (pass/fail) is pass, copies the contents of the stack area to the buffer RAM. It then tests the stack area using the March X algorithm, restores the contents saved in the buffer RAM and the stack pointer (SP). It then returns the test result (pass/fail). This module runs after the application system is initialized. The function RAM_Test_Failure is called by the test harness control file (main.c) when an error is detected.	
Input Parameters	
STACK_TEST_AREA *p	A pointer to a structure that stores the buffer RAM, its size, and the new stack area.
Output Parameters	
NONE	N/A
Return Values	
char	Test result 0 = Test passed. 1 = The test or parameter check failed.

Source file: Stack area test parameter structure

Syntax	
static STACK_TEST_AREA stack_test;	
Description	
A structure declaration and instance that provides the parameters passed to the stack area test module (stl_RL78_RamTest_Stacks_x.asm) from the calling function in main.c. Note: The structure is same as that of as the function stl_RL78_RamTest_Stacks_c and the function stl_RL78_RamTest_Stacks_x.	
Input Parameters	
char *pWork;	The top address of the area where the stack contents are saved.
unsigned short length	Test target size
unsigned short offset	The stack area to be tested (offset from the stack TOP)
char *pNewSp	A stack pointer used temporarily during testing
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

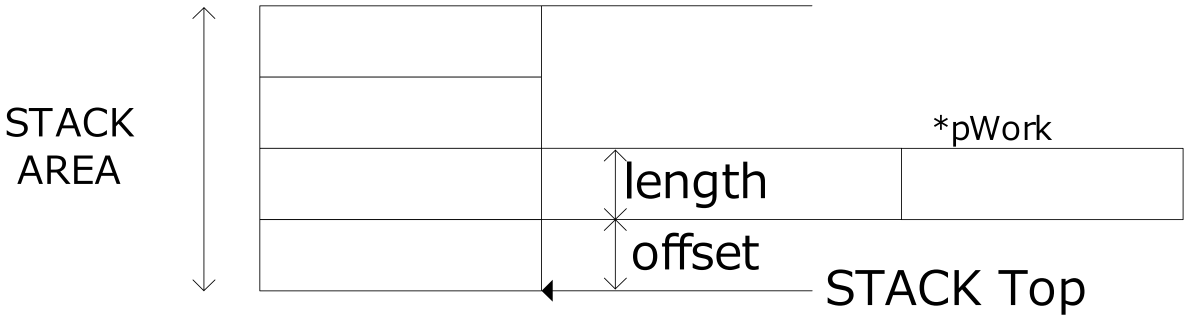


Figure 2-7 The description of Stack area test parameter structure member variable

2.4 System Clock Test

The RL78 self-test library provides self-test modules for testing the internal system clocks (CPU clock and peripheral clock). These modules are used by applications to detect normal and abnormal operation of the main system clocks while the application is running. However, please note that when measuring with the internal low-speed oscillator, the measurement accuracy of the system clock is reduced due to the large error. Therefore, although only the relative operation of the system clock can be seen, it is not a problem to confirm that the system clock is operating normally and that its value is within the allowable limits.

The basic operation of these measurement methods is to detect when the main clock is running at a frequency outside a certain range in the system. The measurement accuracy depends on the accuracy of the reference clock source. For example, an external signal input or a 32 KHz crystal will provide a better measurement accuracy of the system clock than the internal slow oscillator, but this requires additional components.

The test result is returned as “pass/fail”. It also has a built-in means to detect “no reference clock”. If an abnormality is detected, a different test result will be returned than the normal test result.

The module checks whether the measured (captured) time value is within a reference window (between upper and lower limits) based on the reference values defined by the user in the header file “stl_RL78_hw_clocktest.inc”. This header file defines the hardware measurement reference values and the capture interrupt port.

2.4.1 Hardware Measurement

All current RL78 devices include an option in the Timer Array Unit (TAU) channel 5 (or channel 1) that provides additional input capture sources that are designed to be able to test the system clock operation. This capture input is selected in the "Safety" register (TIS0 or TIS1). The options are:

The channels that can be selected as input sources for clock testing vary depending on the microcontroller, so please change according to the microcontroller you are using.

Note: STL function using by channel 1 utilizes ELCL logic cell block 1 and connect the low-speed peripheral clock (f_{SXP}) to TAU channel 1. MCU equipped with ELCL are RL78/G23 and RL78/L23.

- The internal Low-speed oscillator (fiL)
- External 32KHz Oscillator (Sub Clock) (fsub)
- External signal input (TIO5 or TIO1)

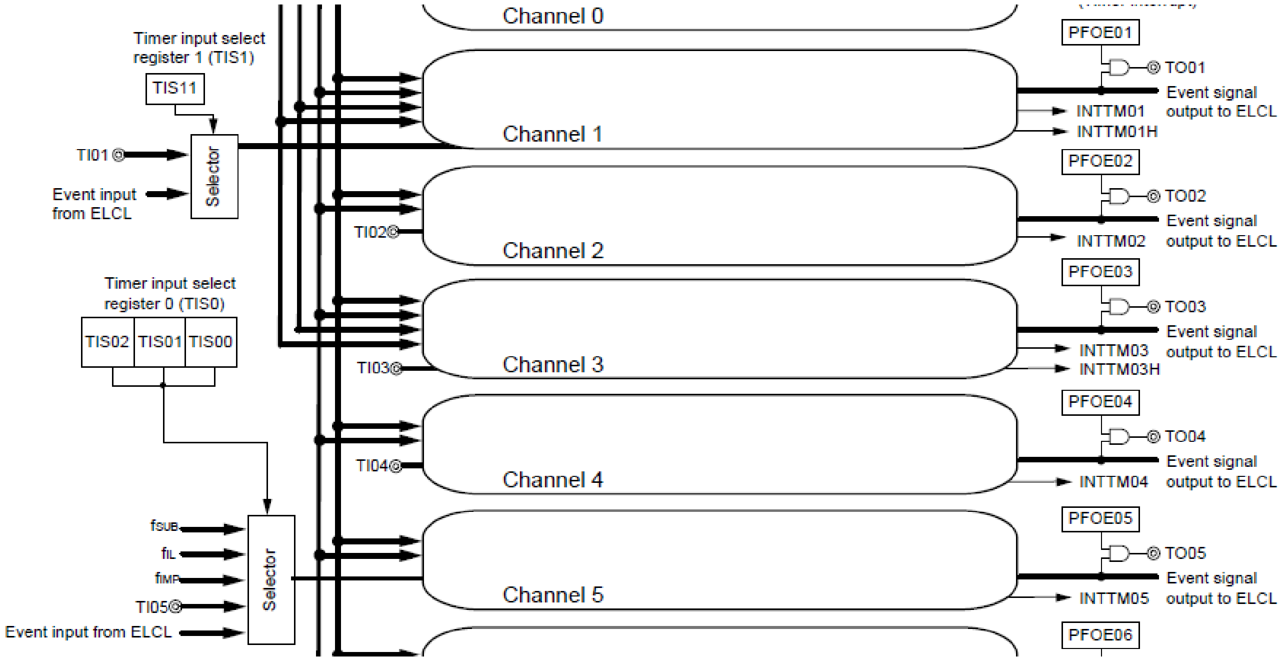


Figure 2-8 Timer Array Unit Channel 1, 5 Configuration

The basic process of hardware measurement is based on the input capture measurement of the reference clock of TAU channel 5. Since it is a hardware capture measurement, the captured time value is a "period" based on the reference clock of the system clock, which is more accurate than software measurement.

The measurement sequence is.

- Synchronise to the reference clock (Wait for first capture event)
- Wait for the next capture event.
- Compare the value in the capture register against the high and lower limit reference values.

The test harness provides an example based on the following settings.

System clock = 32MHz

Reference Clock = 32.768KHz

The calculation is 32000000 / 32768 = 976 (h'3D0)

An allowance should be made for capture value variances in the upper and lower reference values.

Note: RL78/F24 uses timer array unit channel 1.
The reference clock (low-speed on-chip oscillator) of the RL78/F24 is 15KHz.

Table 2-14 Source files: Hardware Clock test

STL FILE NAME	Header Files
stl_RL78_hw_clocktest.asm	stl_RL78_hw_clocktest.inc stl.h
Test Harness File Names	Header Files
main.c	

Syntax	
char stl_RL78_Init_hw_clocktest (unsigned char Select)	
Description	
Initialize hardware measurement (TAU channel 5) for system clock test.	
Input Parameters	
Select	0: TI05 5: fSUB Other: Low-speed on-chip oscillator clock (fIL)
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

Syntax	
char stl_RL78_hw_clocktest(void)	
Description	
It tests the system clock using hardware measurement (TAU channel 5), compares the measurement result (captured value) with the upper and lower limit values defined in the clock test header file (stl_RL78_hw_clocktest.inc), and returns the result (pass/fail/no reference clock) to the calling function. The function Clock_Test_Failure is called by the test harness control file (main.c) when an error is detected.	
Input Parameters	
hwMAXTIME	Upper time limit compare value (defined by stl_RL78_hw_clocktest.inc)
hwMINTIME	Lower time limit compare value (defined by stl_RL78_hw_clocktest.inc)
CAPTURE_interrupt_FLAG	Timer channel Capture Interrupt Flag (defined by stl_RL78_hw_clocktest.inc)
Output Parameters	
NONE	N/A
Return Values	
Char	Test result 0 = Test passed. 1 = The measurement test failed (outside the limits). 2 = The measurement test failed (no reference clock detected).

Table 2-15 Source File: Hardware clock test (ELCL)

STL FILE NAME	Header Files
stl_RL78_hw_clocktestElc.asm	stl_RL78_hw_clocktestElc.inc stl.h
Test Harness File Names	Header Files
main.c	

Syntax	
char stl_RL78_Init_hw_clocktestElc (unsigned char Select)	
Description	
Initialize hardware measurement (TAU channel 1) for system clock test. Select f_{SXP} as the input for cell 0 of logic block 1 in ELCL and specify TAU channel 1 as the output destination.	
Input Parameters	
Select	0: TI01 Other than the above: f_{SXP} Note: f_{SXP} will be either f_{IL} or f_{SUB}
Output Parameters	
None	N/A
Return Values	
None	N/A

Syntax	
char stl_RL78_hw_clocktestElc(void)	
Description	
Test the system clock using hardware measurement (TAU channel 1). Compare the measurement result (captured value) against the upper and lower limits defined in the clock test header file stl_RL78_hw_clocktestElc.inc and return the result (pass/fail/no reference clock) to the calling function. When detecting an anomaly, the test harness control file (main.c) calls the function Clock_Test_Failure to handle the test result.	
Input Parameters	
hwMAXTIME_Elc	Upper limit of comparison (defined by stl_RL78_hw_clocktestElc.inc)
hwMINTIME_Elc	Lower limit of comparison (defined by stl_RL78_hw_clocktestElc.inc)
CAPTURE_interrupt_FLAG_Elc	Timer channel capture interrupt flag (defined by stl_RL78_hw_clocktestElc.inc）
Output Parameters	
None	N/A
Return Values	
char	Test result of CPU register A 0 = Passed 1 = The measurement test failed (outside the limits). 2 = The measurement test failed (no reference clock detected).

2.5 A/D Converter

2.5.1 A/D Converter Test

The RL78/L23 has a function to perform A/D conversion of the positive reference voltage, negative reference voltage, and internal reference voltage. This function can be used to check whether the A/D converter is operating normally. The sample assumes 12-bit resolution.

For detailed A/D conversion settings, please use the Smart Configurator.

Table 2-16 Source files: A/D converter test

STL FILE NAME	Header Files
stl_adc.c	stl_adc.h
Test Harness File Names	Header Files
main.c	

Syntax	
char stl_ADC_Check_TestVoltage (void)	
Description	
The self-diagnosis voltage is defined as the positive reference voltage, negative reference voltage, and internal reference voltage in the header file (stl_adc.h). It compares it against the upper and lower limits specified and returns the result (pass/fail) to the calling function. The function Ad_Test_Failure is called by the test harness control file (main.c) when an error is detected. The calling function must not allow interrupts to occur during this test. Note: A/D conversion is started within the test function. Do not call this function during A/D conversion. Note: The conversion of the internal reference voltage is affected by the voltage of Vref(+). If it is other than 3.3V, please modify the header file (stl_adc.h).	
Input Parameters	
VSS_RANGE_MAX	Upper VSS limit compare value (defined in stl_adc.h)
VDD_RANGE_MIN	Lower VDD limit compare value (defined in stl_adc.h)
AD_RESOLUTION_HEX	Upper VDD limit compare value (defined in stl_adc.h)
VDD	VDD voltage (Vref(+))
VBGR_MIN	Lower internal reference voltage compare value
VBGR_MAX	Upper internal reference voltage compare value
Output Parameters	
NONE	N/A
Return Values	
Char	Test result 0 = Test passed. 1 = The measurement test failed (outside the limits).

2.6 Digital Output

The RL78/L23 have a function that allows you to read the digital output level of a pin when the pin is in output mode. The function can be used to check whether digital output is operating normally.

The ports to be tested are defined in stl_RL78_GpioTest.inc.

Table 2-17 Source files: Digital output test

STL FILE NAME	Header Files
stl_RL78_GpioTest.asm	stl_RL78_GpioTest.inc stl.h
Test Harness File Names	Header Files
main.c	

Syntax	
char stl_RL78_GpioTest (void)	
Description	
This function outputs 0 or 1 based on the static variable (TEST_DATA). The output value and pin level are compared, and the result is returned to the original calling function. The function Gpio_Test_Failure is called by the test harness control file (main.c) when an error is detected. ※ Please set the test port as an output port in the Smart Configurator.	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
Char	Test result 0 = Test passed. 1 = The test or parameter check failed.

2.7 Watchdog

The Watchdog is used to detect abnormal program execution. If the program is not running as expected, an error is detected because the watchdog is not refreshed by software when needed.

The RL78/L23 watchdog timer (WDT) module is used for this purpose. Rather than refreshing just before the specified period, the WDT has a "window" function that always carries out a refresh within the specified "window". The user can set the program so that when an error is detected, an internal reset is generated. A function is provided for use after a reset to determine if a reset was caused by the WDT.

Watchdog values are set in Option Bytes (000C0H/Swap destination address).

Address：000C0H/Swap destination address

<7>	<6>	<5>	<4>	<3>	<2>	<1>	<0>
WDTINT	WINDOW1	WINDOW0	WDTON	WDCS2	WDCS1	WDCS0	WDSTBYON

WDTINT	Use of interval interrupt of watchdog timer
0	Interval interrupt is not used.
1	Interval interrupt is generated when 75% + 1/2 fIL of the overflow time is reached.

WINDOW1	WINDOW0	Watchdog timer window open period
0	0	Setting prohibited
0	1	50%
1	0	75% Setting is prohibited in RL78/L23.
1	**1**	**100%**

WDTON	Operation control of watchdog timer counter
0	Counter operation disabled (counting stopped after reset)
1	**Counter operation enabled (counting started after reset)**

WDCS2	WDCS1	WDCS0	Watchdog timer overflow time (fIL = 37.683 kHz (max.))
0	0	0	2^7/fIL (3.39 ms)
0	0	1	2^8/fIL (6.79 ms)
0	1	0	2^9/fIL (13.58 ms)
0	1	1	2^{10}/fIL (27.17 ms)
1	**0**	**0**	**2^{12}/fIL (108.69 ms)**
1	0	1	2^{14}/fIL (434.78 ms)
1	1	0	2^{15}/fIL (869.56 ms)
1	1	1	2^{17}/fIL (3478.26 ms)

WDSTBYON	Operation control of watchdog timer counter (HALT/STOP mode)
0	Counter operation stopped in HALT/STOP mode
1	**Counter operation enabled in HALT/STOP mode**

Table 2-18 Source files: Watchdog timer test

STL FILE NAME	Header Files
Config_WDT.c	Config_WDT.h
Test Harness File Names	Header Files
main.c	

Syntax	
void R_Config_WDT_Restart (void)	
Description	
Refresh the watchdog count Note: This function was automatically generated by the Smart Configurator.	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

3. Example Usage

In addition to the actual test software source files, the CS+ test harness workspace is provided which includes application examples demonstrating how the tests can be run. This code should be examined in conjunction with this document to see how the various test functions are used.

The testing can be split into two parts:

(1) Power-on Tests.

These tests can be run following a power on or reset. They should be run as soon as possible to ensure that the system is working correctly. These tests are

- All RAM test using Initial March C- (or initial March X)
- All register tests
- Flash Memory CRC Test

The clock tests can also be run later if the initial clock speed values are set to measure the maximum clock speed.

(2) Periodic Tests.

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 run. How the scheduling of the periodic tests is performed is up to the user depending upon how their application is structured.

- RAM tests

  These tests should use the “system” RAM test modules as these are designed to test the memory in small once the system is initialized. They can be used in small in order to minimize the size of the buffer area needed to save the application data.

- Register Tests.

  These are dependent upon the application timing.

- Flash memory test.

  These modules are designed to be able to accumulate a CRC result over a few passes. In this way they can be used to suit the system operation.

The clock test modules can be run at any time to suit the application timing.

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

3.1 CPU

Any CPU test fault is very serious. Therefore, this function’s aim is to move as quickly as possible to a safe position where execution reliability is not relevant.

3.1.1 Power-on Tests

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

3.1.2 Periodic Tests

If testing the CPU registers periodically the function is designed to be run independently and so can be operated at any time to suit the application. Each function restores the original register data on completion of test so as not to corrupt the operation of the application system. It is important that interrupts are disabled during these tests.

3.2 Flash ROM

The ROM test calculates the CRC value of the contents of an area of Flash memory and checks whether a It is compared with a pre-stored reference CRC value.

The CS+ tool chain can calculate and accumulate CRC values and store them in a location specified by the user. In this case, specify "general-purpose CRC" in CS+. See “**Figure 4-5 CS+ Object Conversion Options**”, “**Figure 4-14 CRC Calculation Setting**”.

3.2.1 Power-on Tests

All ROMs used should be tested at power-on. The hardware CRC module It is possible to calculate the CRC value for the entire range of the memory.

3.2.2 Periodic Tests

It is recommended that periodic tests of flash memory be performed in multiple batches while the application is free. Moreover, when using a software module, the application must save the partial calculation result, which will be used as the initial value for the next CRC calculation.

When using a hardware peripheral unit, the results of the partial CRC calculations can be left in the hardware CRC peripheral unit result registers, but it is recommended that they be stored separately and compared before performing the next calculation.

This method allows testing of all Flash memory using time slots convenient for the application.

3.3 RAM Verification

When verifying the RAM, it is important to remember the following points:

- RAM being tested cannot be used for anything else including the current stack.
- Any test requires a RAM buffer where memory contents can be safely copied to and restored from.
- Copy / test / restore the stack area by specifying the backup area and the stack area to be used during the test period. However, interrupt processing cannot be performed during this operation.

3.3.1 Power-on Tests

It is recommended that an initial RAM test module (March C- or March X) be used. These modules are designed specifically for testing all RAM areas at power-up or reset. They are also suitable for execution before initializing the system and the C stack, since they do not require function calls but destroy the contents of the RAM. In this library, the initial RAM test call is implemented in the assembler file ‘cstart.asm’.

3.3.2 Periodic Tests

It is recommended that periodic testing of RAM be performed multiple times, usually using free time in the application. In addition, an area is required to temporarily store the contents of the RAM during the test. Each test will inform you of the pass/fail results for the specified range. This allows all RAM to be verified using a time slot convenient to the application.

3.4 System Clock

It is very serious if a fault is detected with the system clock. The aim of this test should be to get to a safe operating point, where system can be controlled using a different known clock.

3.4.1 Power-on Tests

The system clock should be tested at power-up or reset. The clock should also be tested when the system is initialized and when the system clock frequency is fully set and operation is stable.

3.4.2 Periodic Tests

Periodic testing of the system clock can be made at any time where the application has the time available. This is because the reference clock is typically much slower than the system clock to increase the accuracy of the clock measurement.

(System Clock = 32 MHz, Reference Clock = 32.768 KHz)

3.5 A/D Converter

3.5.1 Power-on Tests

The ADC module can be tested using the “stl_ADC_Check_TestVoltage” function at power-up in the same manner as the periodic test. This function carries out A/D conversion on either the positive reference voltage, negative reference voltage, or internal reference voltage.

3.5.2 Periodic Tests

The “stl_ADC_Check_TestVoltage” function must be called up regularly to perform periodic tests. The reference voltage switched between the negative, positive and internal reference voltages.

3.6 Digital Output

3.6.1 Power-on Tests

The digital output can be tested using the “stl_RL78_GpioTest” function at power-up in the same manner as the periodic test. This function checks whether the output value is 0 or 1.

3.6.2 Periodic Tests

The “stl_RL78_GpioTest” function must be called up regularly to perform periodic tests. Output switches between 0 and 1.

3.7 Watchdog

The watchdog timer function is set in Option Bytes (000C0H/Swap destination address). Following reset release, the watchdog timer starts the count operation. After this, the watchdog needs to be refreshed on a regular basis to prevent time out and reset. Note that, when using the window function, not only does the watchdog need to be regularly refreshed, but it must also be conducted within a time period that matches the specified window. Watchdog refresh is executed by calling the following function.

```
/* Periodic refresh of watchdog to prevent reset from occurring*/

R_Config_WDT_Restart ();
```

If the watchdog is configured to generate a reset when an error is detected, the user program needs to process the interrupt generated by the reset. The sample program is configured to be called the Watchdog_Test_Failure function when an error is detected.

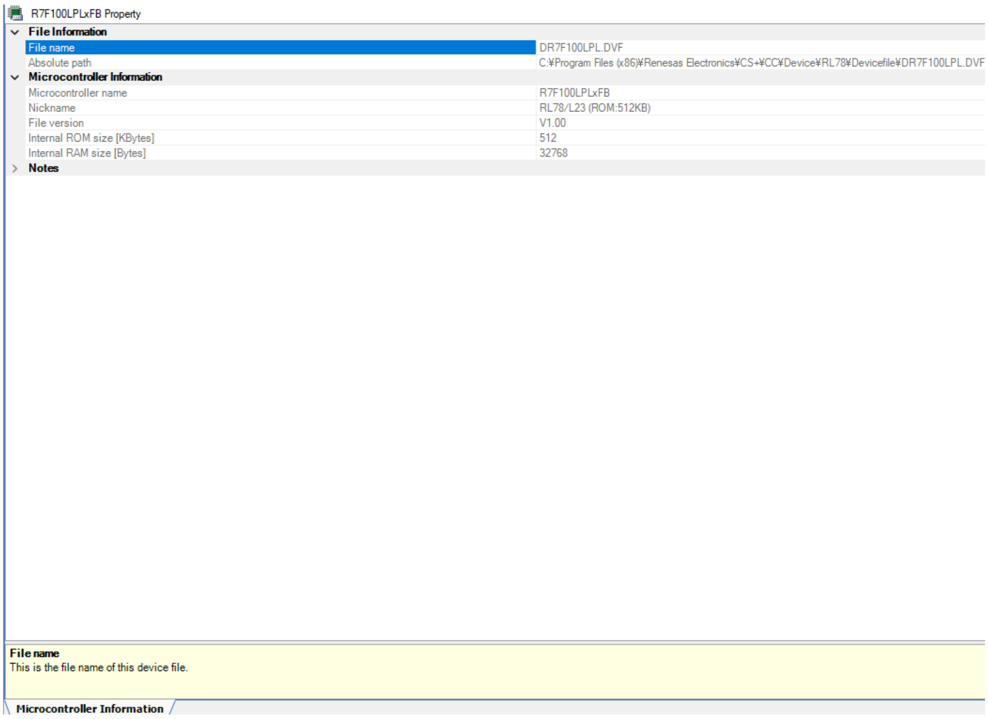
4. Development Environment

E2-Lite	On-chip debugging emulator	
RL78/L23 Fast Prototyping Board	RL78/L23 (100pin LFQFP)	
Tool chain	CS+ Version 8.14.0.00 CC-RL	Version 1.15.1
	e2Studio 2025-07 CC-RL	Version 1.15.1
MCU	R7F100LPL3CFB	
Internal Clock	32 MHz High-speed On-Chip Oscillator	
System clock	32 MHz	
Low speed clock	32.768kHz Low-speed On-Chip Oscillator	

4.1 CS+ Settings

The following show the specific options and setting set for the test project. The graphics only show those options and settings that have been changed. All others are the default project settings set by the CS+.

4.1.1 Common Options

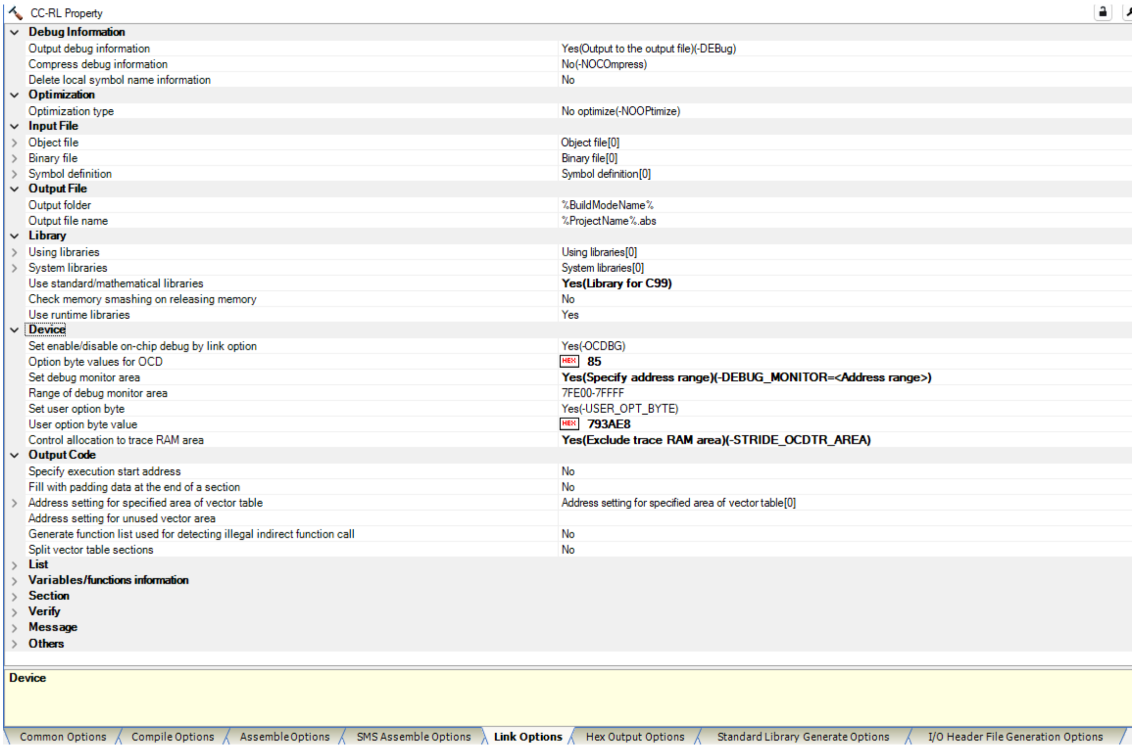


R7F100LPLxFB Property

File Information	
File name	DR7F100LPL.DVF
Absolute path	C:¥Program Files (x86)¥Renesas Electronics¥CS+¥CC¥Device¥RL78¥Devicefile¥DR7F100LPL.DVF
Microcontroller Information	
Microcontroller name	R7F100LPLxFB
Nickname	RL78/L23 (ROM:512KB)
File version	V1.00
Internal ROM size [KBytes]	512
Internal RAM size [Bytes]	32768
Notes	

File name
This is the file name of this device file.

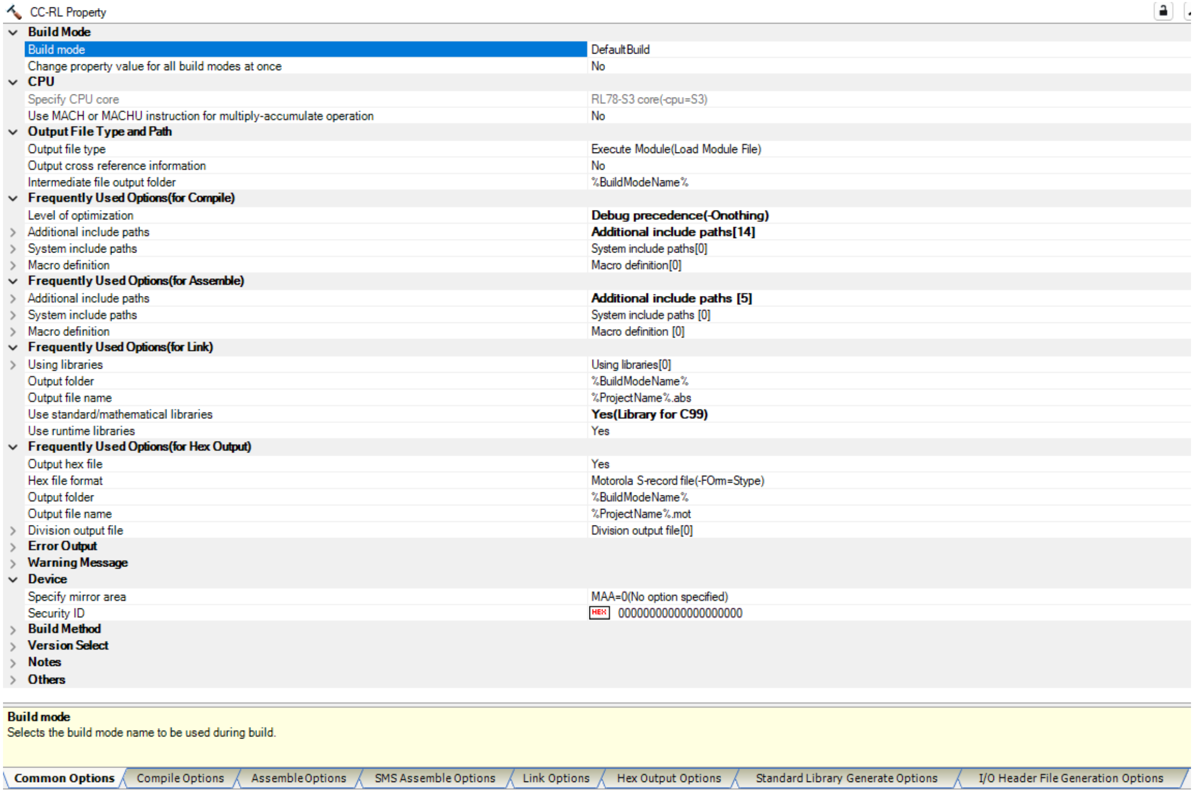
Figure 4-1 CS+ common Options - Target Device



Debug Information	
Output debug information	Yes(Output to the output file)(-DEBug)
Compress debug information	No(-NOCOmpress)
Delete local symbol name information	No
Optimization	
Optimization type	No optimize(-NOOPtimize)
Input File	
Object file	Object file[0]
Binary file	Binary file[0]
Symbol definition	Symbol definition[0]
Output File	
Output folder	%BuildModeName%
Output file name	%ProjectName%.abs
Library	
Using libraries	Using libraries[0]
System libraries	System libraries[0]
Use standard/mathematical libraries	**Yes(Library for C99)**
Check memory smashing on releasing memory	No
Use runtime libraries	Yes
Device	
Set enable/disable on-chip debug by link option	Yes(-OCDBG)
Option byte values for OCD	HEX **85**
Set debug monitor area	**Yes(Specify address range)(-DEBUG_MONITOR=<Address range>)**
Range of debug monitor area	7FE00-7FFFF
Set user option byte	Yes(-USER_OPT_BYTE)
User option byte value	HEX **793AE8**
Control allocation to trace RAM area	**Yes(Exclude trace RAM area)(-STRIDE_OCDTR_AREA)**
Output Code	
Specify execution start address	No
Fill with padding data at the end of a section	No
Address setting for specified area of vector table	Address setting for specified area of vector table[0]
Address setting for unused vector area	
Generate function list used for detecting illegal indirect function call	No
Split vector table sections	No
List	
Variables/functions information	
Section	
Verify	
Message	
Others	

Device

Figure 4-2 CS+ Link Options

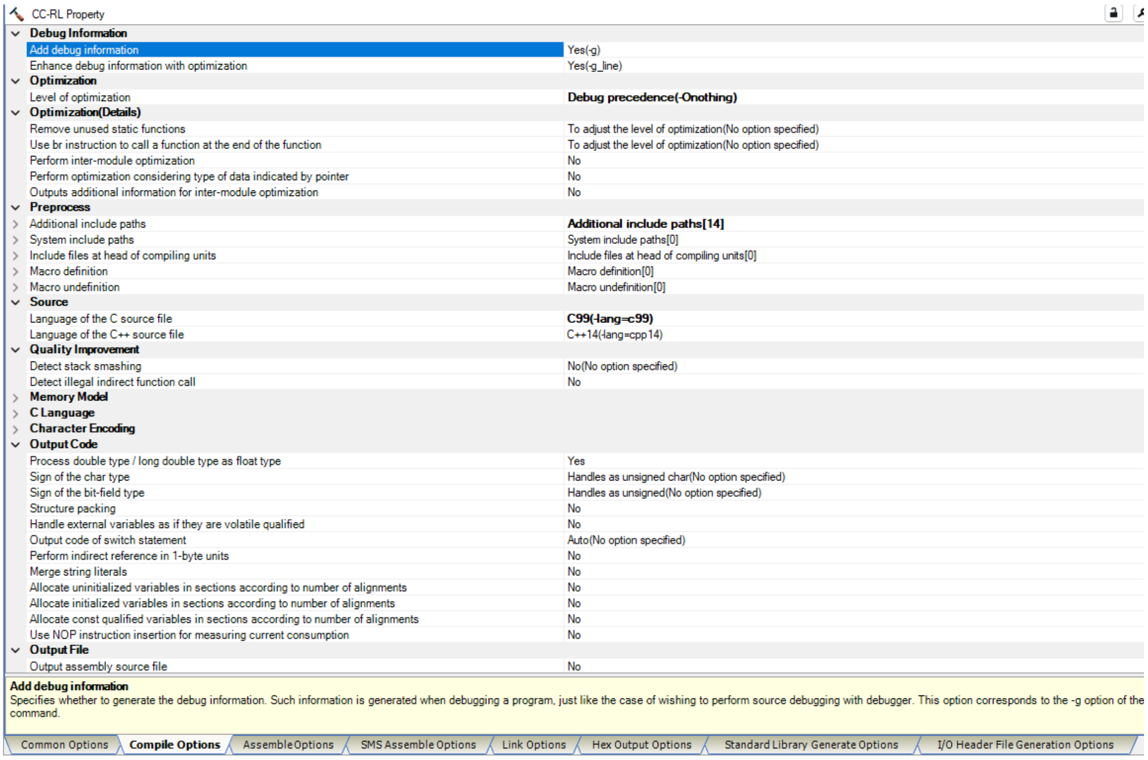


Build Mode	
Build mode	DefaultBuild
Change property value for all build modes at once	No
CPU	
Specify CPU core	RL78-S3 core(-cpu=S3)
Use MACH or MACHU instruction for multiply-accumulate operation	No
Output File Type and Path	
Output file type	Execute Module(Load Module File)
Output cross reference information	No
Intermediate file output folder	%BuildModeName%
Frequently Used Options(for Compile)	
Level of optimization	**Debug precedence(-Onothing)**
Additional include paths	**Additional include paths[14]**
System include paths	System include paths[0]
Macro definition	Macro definition[0]
Frequently Used Options(for Assemble)	
Additional include paths	**Additional include paths [5]**
System include paths	System include paths [0]
Macro definition	Macro definition [0]
Frequently Used Options(for Link)	
Using libraries	Using libraries[0]
Output folder	%BuildModeName%
Output file name	%ProjectName%.abs
Use standard/mathematical libraries	**Yes(Library for C99)**
Use runtime libraries	Yes
Frequently Used Options(for Hex Output)	
Output hex file	Yes
Hex file format	Motorola S-record file(-FOrm=Stype)
Output folder	%BuildModeName%
Output file name	%ProjectName%.mot
Division output file	Division output file[0]
Error Output	
Warning Message	
Device	
Specify mirror area	MAA=0(No option specified)
Security ID	HEX 00000000000000000000
Build Method	
Version Select	
Notes	
Others	

Build mode
Selects the build mode name to be used during build.

Figure 4-3 CS+ Common options

4.1.2 Complier Setting

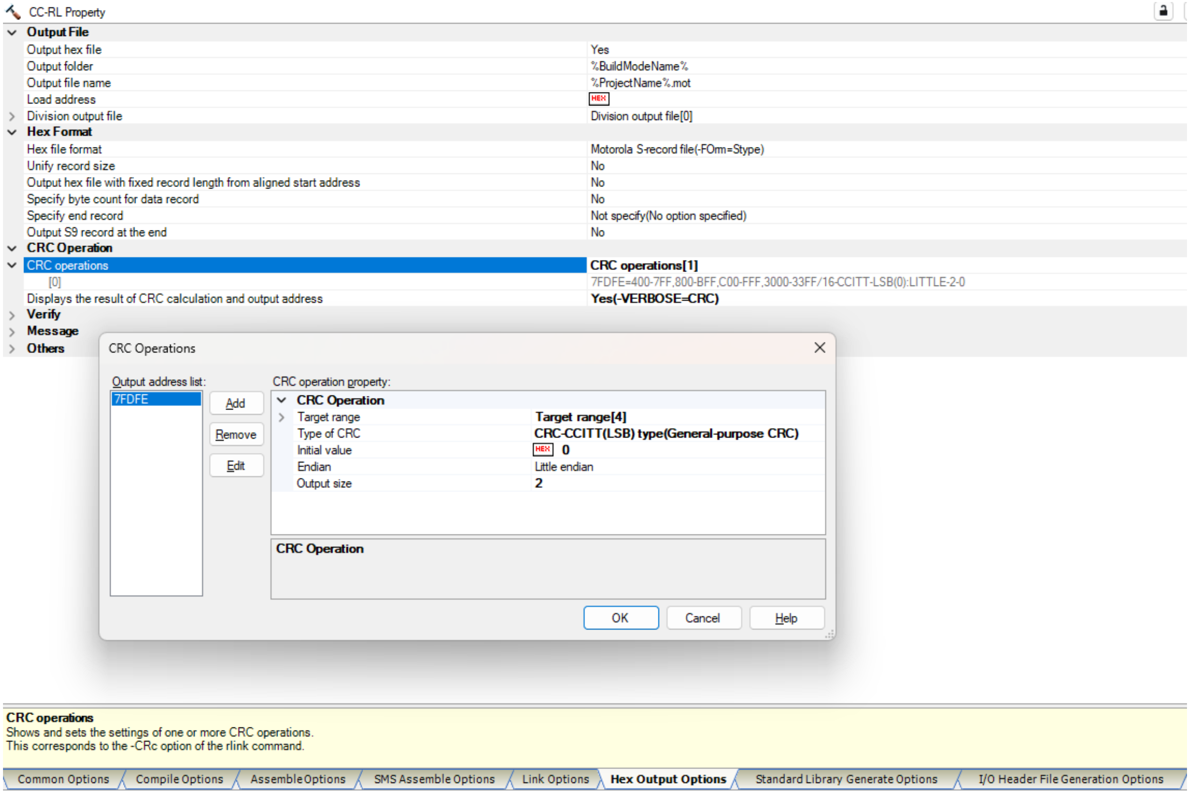


CC-RL Property

Debug Information	
Add debug information	Yes(-g)
Enhance debug information with optimization	Yes(-g_line)
Optimization	
Level of optimization	**Debug precedence(-Onothing)**
Optimization(Details)	
Remove unused static functions	To adjust the level of optimization(No option specified)
Use br instruction to call a function at the end of the function	To adjust the level of optimization(No option specified)
Perform inter-module optimization	No
Perform optimization considering type of data indicated by pointer	No
Outputs additional information for inter-module optimization	No
Preprocess	
Additional include paths	**Additional include paths[14]**
System include paths	System include paths[0]
Include files at head of compiling units	Include files at head of compiling units[0]
Macro definition	Macro definition[0]
Macro undefinition	Macro undefinition[0]
Source	
Language of the C source file	**C99(-lang=c99)**
Language of the C++ source file	C++14(-lang=cpp14)
Quality Improvement	
Detect stack smashing	No(No option specified)
Detect illegal indirect function call	No
Memory Model	
C Language	
Character Encoding	
Output Code	
Process double type / long double type as float type	Yes
Sign of the char type	Handles as unsigned char(No option specified)
Sign of the bit-field type	Handles as unsigned(No option specified)
Structure packing	No
Handle external variables as if they are volatile qualified	No
Output code of switch statement	Auto(No option specified)
Perform indirect reference in 1-byte units	No
Merge string literals	No
Allocate uninitialized variables in sections according to number of alignments	No
Allocate initialized variables in sections according to number of alignments	No
Allocate const qualified variables in sections according to number of alignments	No
Use NOP instruction insertion for measuring current consumption	No
Output File	
Output assembly source file	No

Add debug information
Specifies whether to generate the debug information. Such information is generated when debugging a program, just like the case of wishing to perform source debugging with debugger. This option corresponds to the -g option of the command.

Figure 4-4 CS+ Compiler Options



CC-RL Property

Output File	
Output hex file	Yes
Output folder	%BuildModeName%
Output file name	%ProjectName%.mot
Load address	HEX
Division output file	Division output file[0]
Hex Format	
Hex file format	Motorola S-record file(-FOrm=Stype)
Unify record size	No
Output hex file with fixed record length from aligned start address	No
Specify byte count for data record	No
Specify end record	Not specify(No option specified)
Output S9 record at the end	No
CRC Operation	
CRC operations	**CRC operations[1]**
[0]	7FDFE=400-7FF,800-BFF,C00-FFF,3000-33FF/16-CCITT-LSB(0):LITTLE-2-0
Displays the result of CRC calculation and output address	**Yes(-VERBOSE=CRC)**
Verify	
Message	
Others	

CRC Operations

Output address list:
7FDFE

CRC operation property:

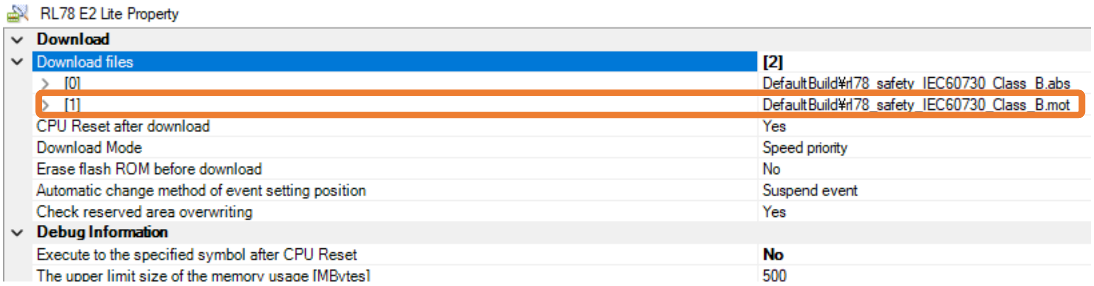
CRC Operation	
Target range	**Target range[4]**
Type of CRC	**CRC-CCITT(LSB) type(General-purpose CRC)**
Initial value	HEX **0**
Endian	Little endian
Output size	**2**

CRC Operation

CRC operations
Shows and sets the settings of one or more CRC operations.
This corresponds to the -CRc option of the rlink command.

Figure 4-5 CS+ Object Conversion Options

4.1.3 Download file of debug tool setting

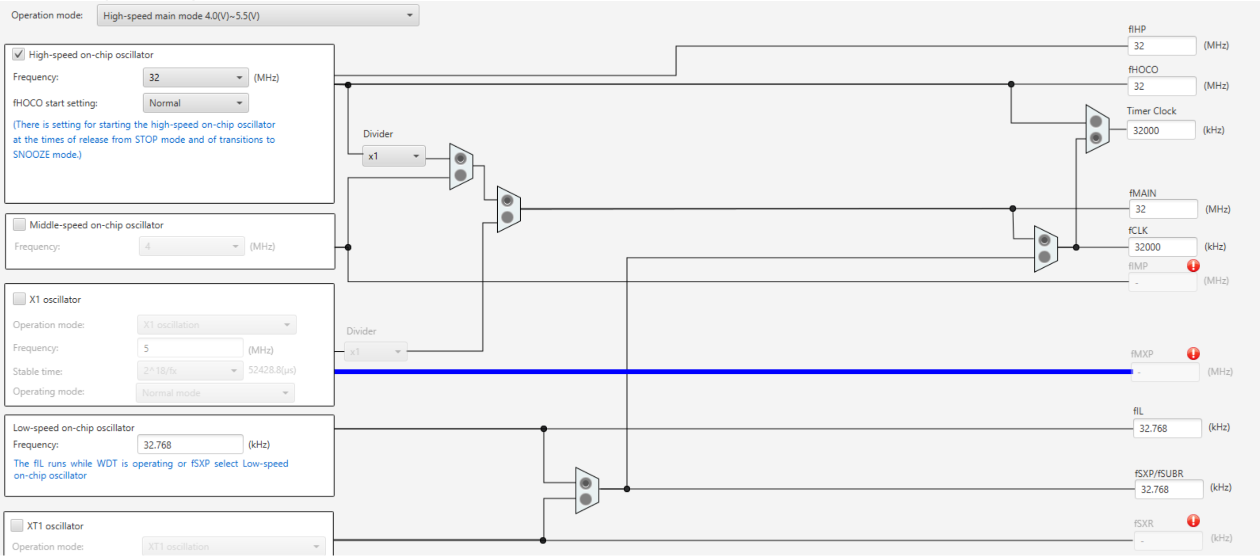


RL78 E2 Lite Property

Download	
Download files	**[2]**
[0]	DefaultBuild¥rl78_safety_IEC60730_Class_B.abs
[1]	DefaultBuild¥rl78_safety_IEC60730_Class_B.mot
CPU Reset after download	Yes
Download Mode	Speed priority
Erase flash ROM before download	No
Automatic change method of event setting position	Suspend event
Check reserved area overwriting	Yes
Debug Information	
Execute to the specified symbol after CPU Reset	**No**
The upper limit size of the memory usage [MBytes]	500

Figure 4-6 Download file of CS+ debug tool setting

4.1.4 Code Generation (design tool)



Operation mode: High-speed main mode 4.0(V)~5.5(V)

☑ High-speed on-chip oscillator
Frequency: 32 (MHz)
fHOCO start setting: Normal
(There is setting for starting the high-speed on-chip oscillator at the times of release from STOP mode and of transitions to SNOOZE mode.)

□ Middle-speed on-chip oscillator
Frequency: 4 (MHz)

□ X1 oscillator
Operation mode: X1 oscillation
Frequency: 5 (MHz)
Stable time: 2^18/fx 52428.8(μs)
Operating mode: Normal mode

Low-speed on-chip oscillator
Frequency: 32.768 (kHz)
The fIL runs while WDT is operating or fSXP select Low-speed on-chip oscillator

□ XT1 oscillator
Operation mode: XT1 oscillation

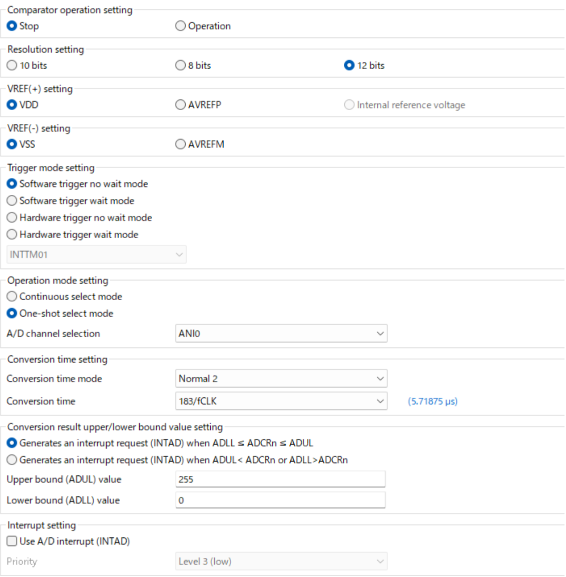
Divider: x1

Divider: x1

fIHP: 32 (MHz)
fHOCO: 32 (MHz)
Timer Clock: 32000 (kHz)
fMAIN: 32 (MHz)
fCLK: 32000 (kHz)
fIMP: - (MHz)
fMXP: - (MHz)
fIL: 32.768 (kHz)
fSXP/fSUBR: 32.768 (kHz)
fSXR: - (kHz)

Figure 4-7 Clock Setting

4.1.4.2 A/D converter



Comparator operation setting
● Stop ○ Operation

Resolution setting
○ 10 bits ○ 8 bits ● 12 bits

VREF(+) setting
● VDD ○ AVREFP ○ Internal reference voltage

VREF(-) setting
● VSS ○ AVREFM

Trigger mode setting
● Software trigger no wait mode
○ Software trigger wait mode
○ Hardware trigger no wait mode
○ Hardware trigger wait mode
INTTM01

Operation mode setting
○ Continuous select mode
● One-shot select mode
A/D channel selection: ANI0

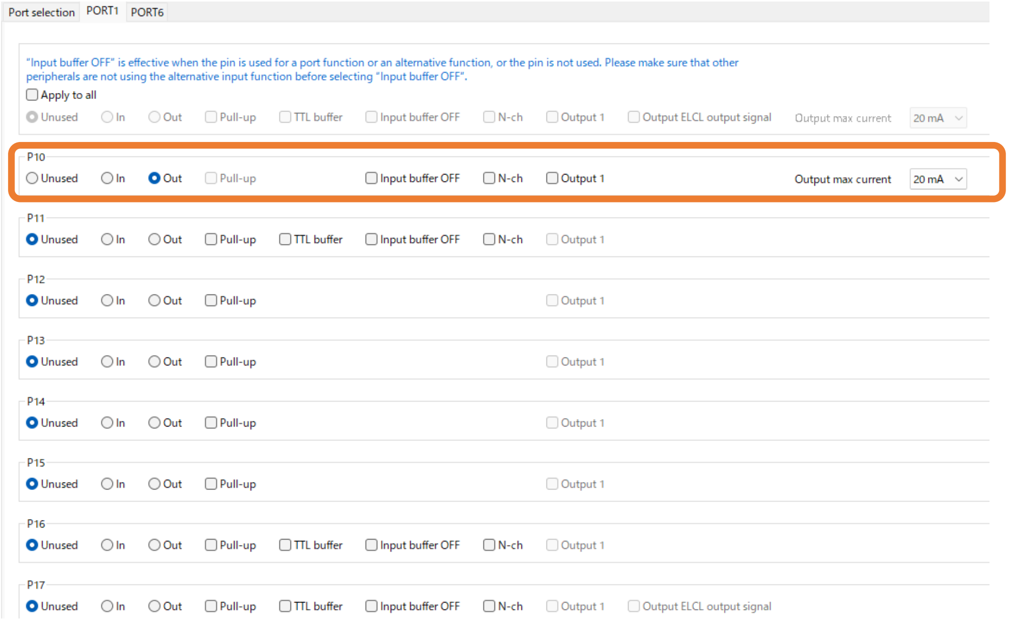
Conversion time setting
Conversion time mode: Normal 2
Conversion time: 183/fCLK (5.71875 μs)

Conversion result upper/lower bound value setting
● Generates an interrupt request (INTAD) when ADLL ≤ ADCRn ≤ ADUL
○ Generates an interrupt request (INTAD) when ADUL < ADCRn or ADLL > ADCRn
Upper bound (ADUL) value: 255
Lower bound (ADLL) value: 0

Interrupt setting
□ Use A/D interrupt (INTAD)
Priority: Level 3 (low)

Figure 4-8 A/D Convertor Setting

4.1.4.3 PORT setting



Port selection | PORT1 | PORT6

"Input buffer OFF" is effective when the pin is used for a port function or an alternative function, or the pin is not used. Please make sure that other peripherals are not using the alternative input function before selecting "Input buffer OFF".

□ Apply to all

● Unused ○ In ○ Out □ Pull-up □ TTL buffer □ Input buffer OFF □ N-ch □ Output 1 □ Output ELCL output signal Output max current 20 mA

P10
○ Unused ○ In ● Out □ Pull-up □ Input buffer OFF □ N-ch □ Output 1 Output max current 20 mA

P11
● Unused ○ In ○ Out □ Pull-up □ TTL buffer □ Input buffer OFF □ N-ch □ Output 1

P12
● Unused ○ In ○ Out □ Pull-up □ Output 1

P13
● Unused ○ In ○ Out □ Pull-up □ Output 1

P14
● Unused ○ In ○ Out □ Pull-up □ Output 1

P15
● Unused ○ In ○ Out □ Pull-up □ Output 1

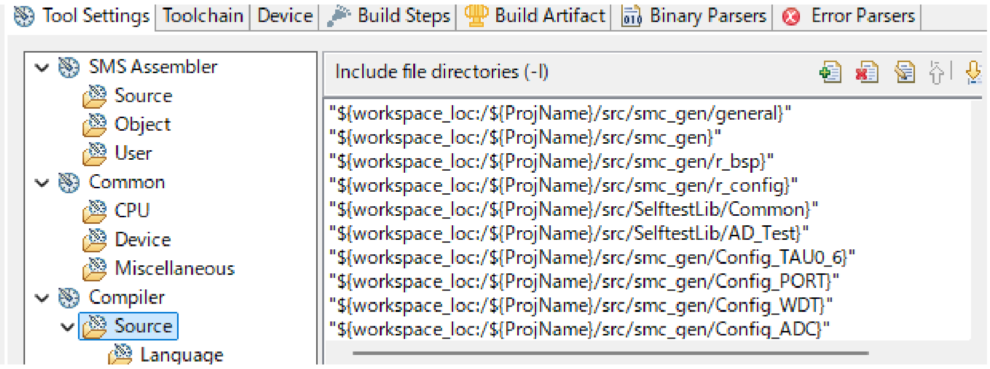
P16
● Unused ○ In ○ Out □ Pull-up □ TTL buffer □ Input buffer OFF □ N-ch □ Output 1

P17
● Unused ○ In ○ Out □ Pull-up □ TTL buffer □ Input buffer OFF □ N-ch □ Output 1 □ Output ELCL output signal

Figure 4-9 Output setting of GPIO test port

4.2 Setting up e^{2}studio

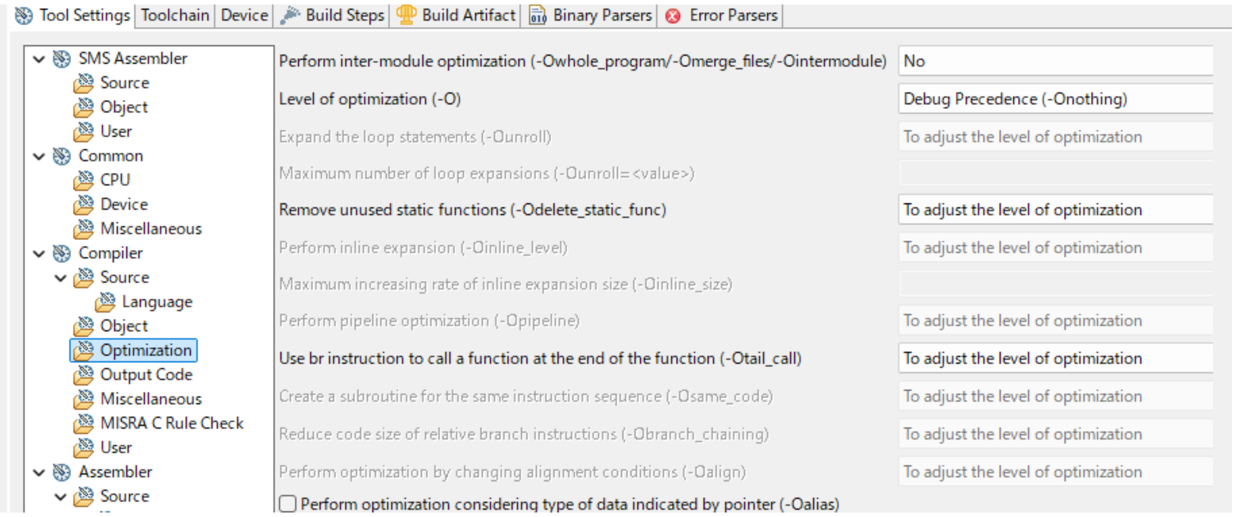
4.2.1 Compiler Options



Include file directories (-I)

```
"${workspace_loc:/${ProjName}/src/smc_gen/general}"
"${workspace_loc:/${ProjName}/src/smc_gen}"
"${workspace_loc:/${ProjName}/src/smc_gen/r_bsp}"
"${workspace_loc:/${ProjName}/src/smc_gen/r_config}"
"${workspace_loc:/${ProjName}/src/SelftestLib/Common}"
"${workspace_loc:/${ProjName}/src/SelftestLib/AD_Test}"
"${workspace_loc:/${ProjName}/src/smc_gen/Config_TAU0_6}"
"${workspace_loc:/${ProjName}/src/smc_gen/Config_PORT}"
"${workspace_loc:/${ProjName}/src/smc_gen/Config_WDT}"
"${workspace_loc:/${ProjName}/src/smc_gen/Config_ADC}"
```

Figure 4-10 C source include path

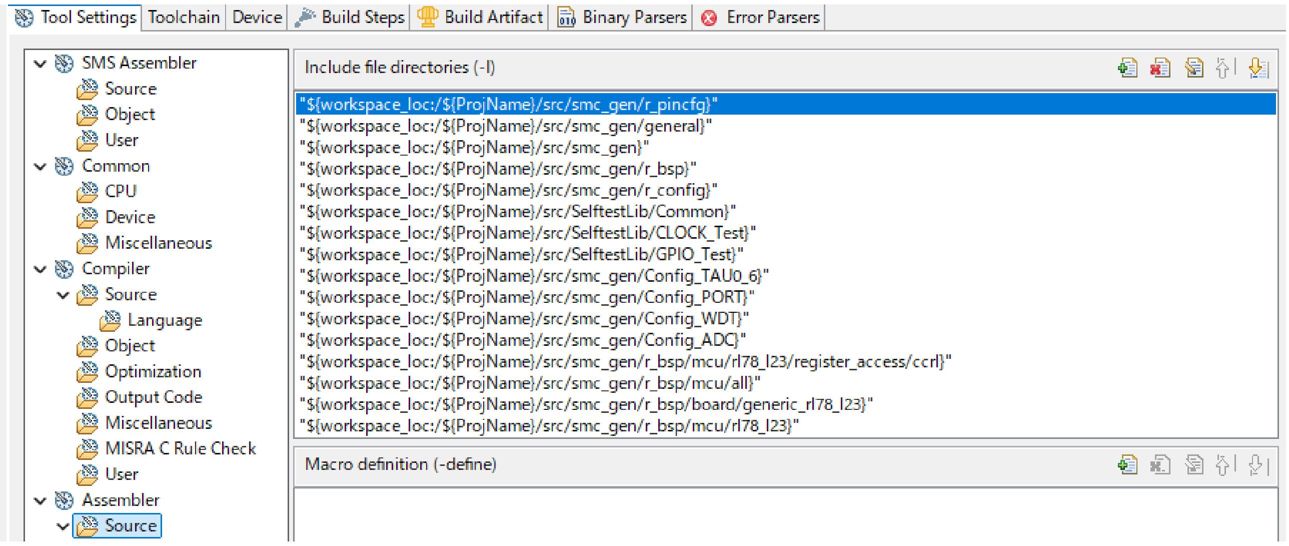


Perform inter-module optimization (-Owhole_program/-Omerge_files/-Ointermodule)	No
Level of optimization (-O)	Debug Precedence (-Onothing)
Expand the loop statements (-Ounroll)	To adjust the level of optimization
Maximum number of loop expansions (-Ounroll=<value>)	
Remove unused static functions (-Odelete_static_func)	To adjust the level of optimization
Perform inline expansion (-Oinline_level)	To adjust the level of optimization
Maximum increasing rate of inline expansion size (-Oinline_size)	
Perform pipeline optimization (-Opipeline)	To adjust the level of optimization
Use br instruction to call a function at the end of the function (-Otail_call)	To adjust the level of optimization
Create a subroutine for the same instruction sequence (-Osame_code)	To adjust the level of optimization
Reduce code size of relative branch instructions (-Obranch_chaining)	To adjust the level of optimization
Perform optimization by changing alignment conditions (-Oalign)	To adjust the level of optimization

☐ Perform optimization considering type of data indicated by pointer (-Oalias)

Figure 4-11 Optimization

4.2.2 Assembler Options



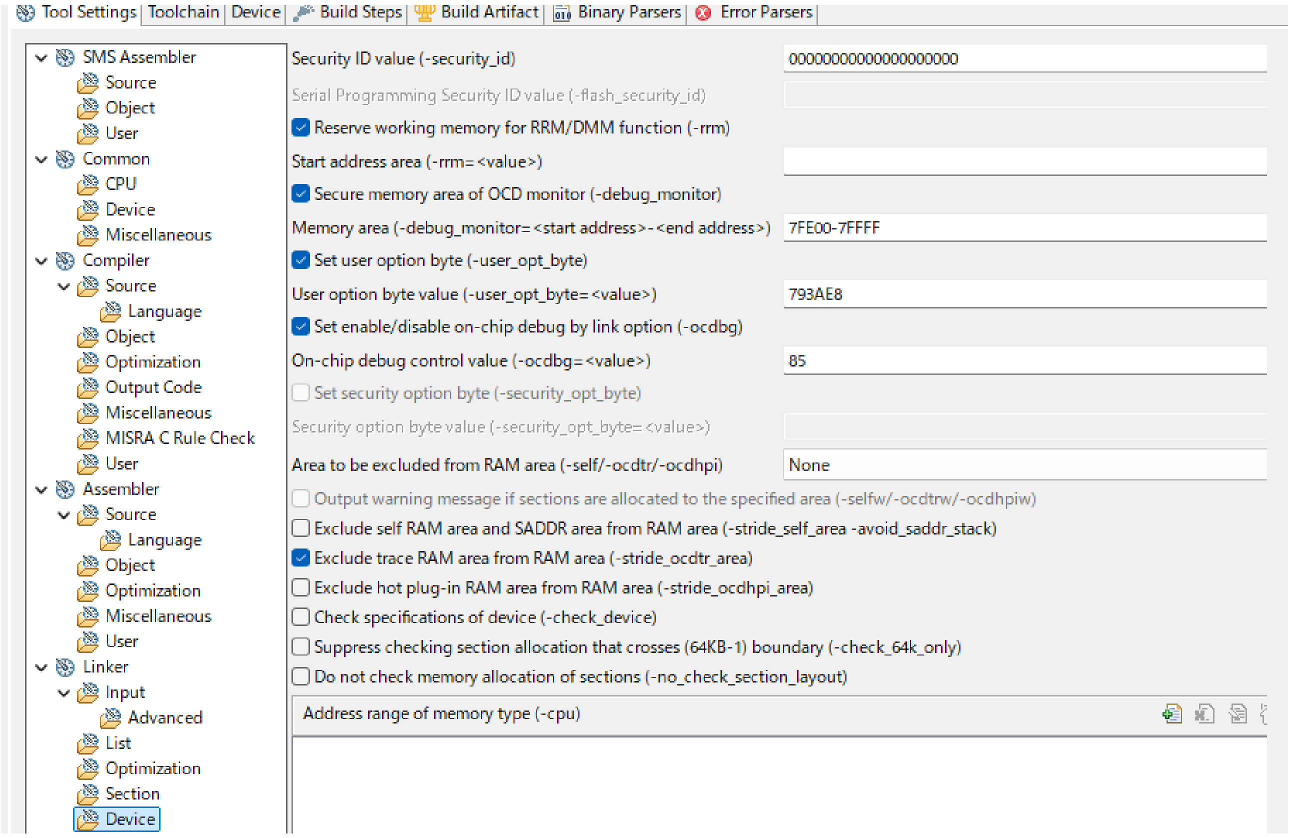
Include file directories (-I)

```
"${workspace_loc:/${ProjName}/src/smc_gen/r_pincfg}"
"${workspace_loc:/${ProjName}/src/smc_gen/general}"
"${workspace_loc:/${ProjName}/src/smc_gen}"
"${workspace_loc:/${ProjName}/src/smc_gen/r_bsp}"
"${workspace_loc:/${ProjName}/src/smc_gen/r_config}"
"${workspace_loc:/${ProjName}/src/SelftestLib/Common}"
"${workspace_loc:/${ProjName}/src/SelftestLib/CLOCK_Test}"
"${workspace_loc:/${ProjName}/src/SelftestLib/GPIO_Test}"
"${workspace_loc:/${ProjName}/src/smc_gen/Config_TAU0_6}"
"${workspace_loc:/${ProjName}/src/smc_gen/Config_PORT}"
"${workspace_loc:/${ProjName}/src/smc_gen/Config_WDT}"
"${workspace_loc:/${ProjName}/src/smc_gen/Config_ADC}"
"${workspace_loc:/${ProjName}/src/smc_gen/r_bsp/mcu/rl78_l23/register_access/ccrl}"
"${workspace_loc:/${ProjName}/src/smc_gen/r_bsp/mcu/all}"
"${workspace_loc:/${ProjName}/src/smc_gen/r_bsp/board/generic_rl78_l23}"
"${workspace_loc:/${ProjName}/src/smc_gen/r_bsp/mcu/rl78_l23}"
```

Macro definition (-define)

Figure 4-12 asm source include path

4.2.3 Linker options



Security ID value (-security_id): 00000000000000000000

Serial Programming Security ID value (-flash_security_id)

☑ Reserve working memory for RRM/DMM function (-rrm)

Start address area (-rrm=<value>)

☑ Secure memory area of OCD monitor (-debug_monitor)

Memory area (-debug_monitor=<start address>-<end address>): 7FE00-7FFFF

☑ Set user option byte (-user_opt_byte)

User option byte value (-user_opt_byte=<value>): 793AE8

☑ Set enable/disable on-chip debug by link option (-ocdbg)

On-chip debug control value (-ocdbg=<value>): 85

□ Set security option byte (-security_opt_byte)

Security option byte value (-security_opt_byte=<value>)

Area to be excluded from RAM area (-self/-ocdtr/-ocdhpi): None

□ Output warning message if sections are allocated to the specified area (-selfw/-ocdtrw/-ocdhpiw)

□ Exclude self RAM area and SADDR area from RAM area (-stride_self_area -avoid_saddr_stack)

☑ Exclude trace RAM area from RAM area (-stride_ocdtr_area)

□ Exclude hot plug-in RAM area from RAM area (-stride_ocdhpi_area)

□ Check specifications of device (-check_device)

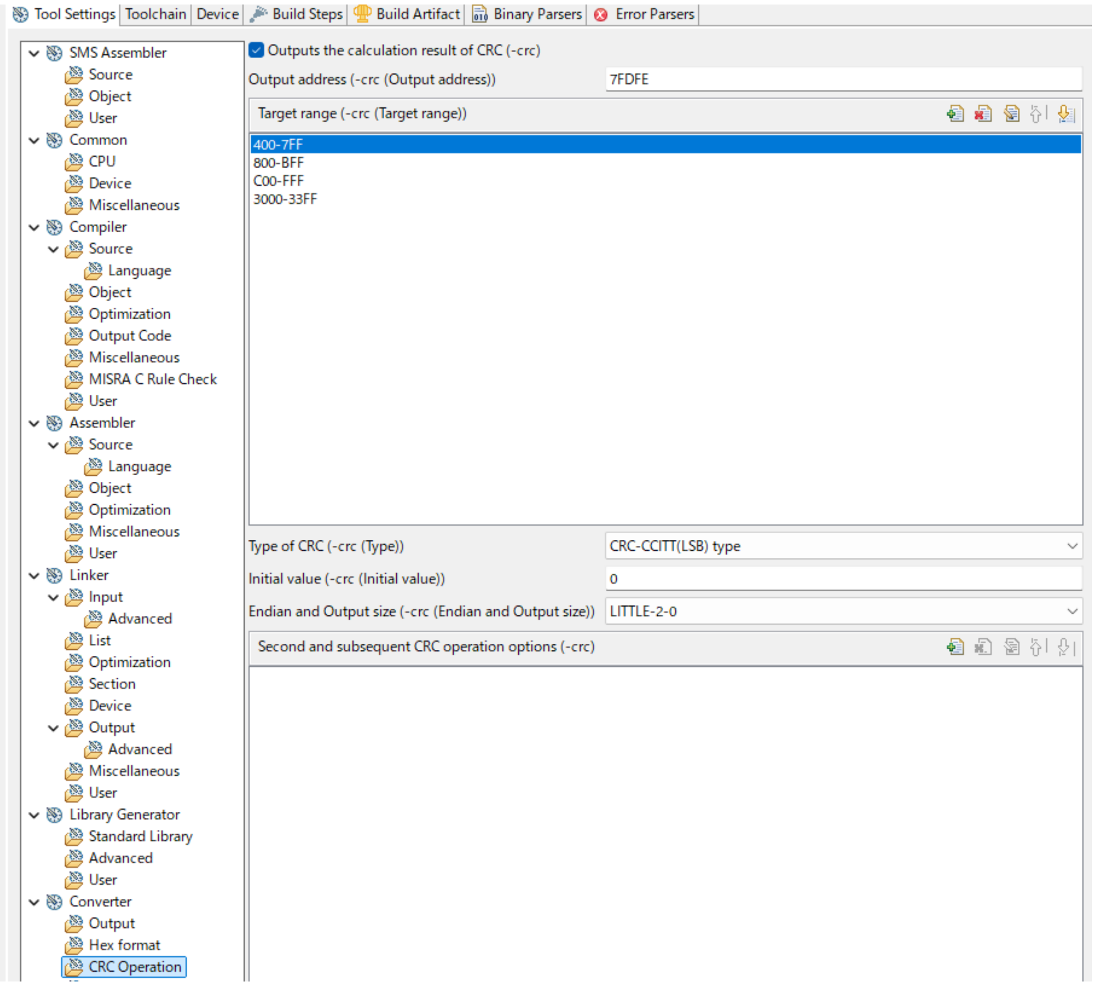
□ Suppress checking section allocation that crosses (64KB-1) boundary (-check_64k_only)

□ Do not check memory allocation of sections (-no_check_section_layout)

Address range of memory type (-cpu)

Figure 4-13 Device Settings

4.2.4 Converter options



☑ Outputs the calculation result of CRC (-crc)

Output address (-crc (Output address)): 7FDFE

Target range (-crc (Target range)):

400-7FF
800-BFF
C00-FFF
3000-33FF

Type of CRC (-crc (Type)): CRC-CCITT(LSB) type

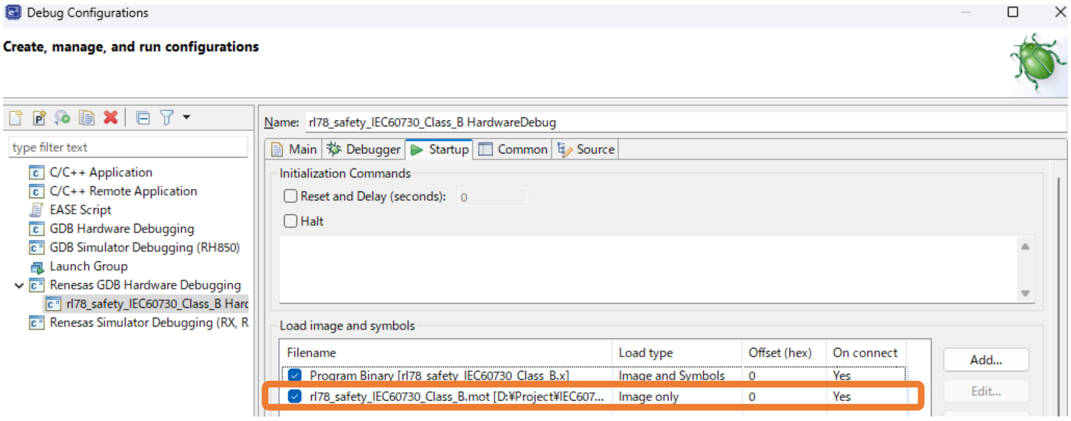
Initial value (-crc (Initial value)): 0

Endian and Output size (-crc (Endian and Output size)): LITTLE-2-0

Second and subsequent CRC operation options (-crc)

Figure 4-14 CRC calculation settings

4.2.5 Debugging Configuration


Debug Configurations
Create, manage, and run configurations
type filter text
C/C++ Application
C/C++ Remote Application
EASE Script
GDB Hardware Debugging
GDB Simulator Debugging (RH850)
Launch Group
Renesas GDB Hardware Debugging
rl78_safety_IEC60730_Class_B Harc
Renesas Simulator Debugging (RX, R
Main
Debugger
Startup
Common
Source
Add...
Edit...

Name: rl78_safety_IEC60730_Class_B HardwareDebug

Initialization Commands

□ Reset and Delay (seconds): 0

□ Halt

Load image and symbols

	Filename	Load type	Offset (hex)	On connect
☑	Program Binary [rl78_safety_IEC60730_Class_B.x]	Image and Symbols	0	Yes
☑	rl78_safety_IEC60730_Class_B.mot [D:¥Project¥IEC607...	Image only	0	Yes

Figure 4-15 Download file for e^{2}studio debug tool configuration

5. Benchmark Test results

Library functions	Number of bytes tested	Processing time
CPU general-purpose register test stl_RL78_RegisterTest	-	10.281µs
CPU register test - PSW stl_RL78_RegisterTest_psw	-	1.343µs
CPU register test - SP stl_RL78_RegisterTest_stack	-	1.125µs
CPU register test - CS stl_RL78_RegisterTest_cs	-	1.031µs
CPU register test - ES stl_RL78_RegisterTest_es	-	1.031µs
CPU register test - PC stl_RL78_RegisterTest_pc	-	0.875µs
Hardware CRC stl_RL78_peripheral_crc	1024 byte	700µs
System March C- stl_RL78_march_c	48 byte	500µs
System March X stl_RL78_march_x	48 byte	300µs
Initial March C- stl_RL78_march_c_initial	1020 byte	11.3ms
Initial March X stl_RL78_march_x_initial	1020 byte	6.3ms
Hardware Clock test stl_RL78_hw_clocktest	-	52.281µs
Hardware Clock test stl_RL78_hw_clocktestElc		52.281µs
Stack area test (March C-) stl_RL78_RamTest_Stacks_c	64byte+64byte	1.4ms
Stack area test (March X) stl_RL78_RamTest_Stacks_x	64byte+64byte	800us
GPIO test stl_RL78_GpioTest	-	1.093µs
A/D test stl_ADC_Check_TestVoltage	-	9.218µs

6. Additional Hardware Resources

The following additional safety and self test features have been included in the RL78 series to provide support for the user. While these additional functions have not been certified by VDE, they provide a valuable extra resource to the user and are included here for reference.

6.1 Additional Safety Functions

The following additional safety functions have been included in the RL78 series MCU devices.

6.1.1 RAM Parity Generator Checker

When enabled the function includes a parity check for each byte written to any location of the RAM memory area. The Parity is generated when data is written to the RAM memory and checked when a location is read from memory.

Note that this function is available only for data access. If a code is executed from RAM, an additional 10 bytes from the area must be initialized. If a RAM parity error is detected, then an internal Reset is generated. The Reset source can be determined by examining the “RESF” register. The “IAWRF” bit will be set if the invalid memory access was the source of the Reset.

Format of RAM Parity Error Control Register (RPECTL)

Address: F00F5H Reset: 00H R/W

Symbol	<7>	6	5	4	3	2	1	<0>
RPECTL	**RPERDIS**	0	0	0	0	0	0	**RPEF**

RPERDIS	Parity error reset mask flag
0	Enables parity error resets.
1	Disables parity error resets.

RPEF	Parity error status flag
0	No parity error has occurred.
1	A parity error has occurred.

Figure 6-1 Checking RAM parity errors

6.1.2 RAM Guard Protection

This is a write protection feature that when enabled allows data to be read from the selected RAM area, but prohibits a write to these locations. No error is generated if a write occurs to this area.

The RAM area available for this feature is limited and can be selected by the “**GRAM0**, **GRAM1**”bits as shown in Figure 6-2 below:

Format of Invalid Memory Access Detection Control Register (IAWCTL)

Address: F0078H Reset: 00H R/W

Symbol	7	6	5	4	3	2	1	0
IAWCTL	IAWEN	0	**GRAM1**	**GRAM0**	0	GPORT	GINT	GCSC

GRAM1	**GRAM0**	Protected area in the RAM
0	0	Disabled. Writing to the RAM is allowed.
0	1	128 bytes from the base address of the RAM
1	0	256 bytes from the base address of the RAM
1	1	512 bytes from the base address of the RAM

Figure 6-2 RAM Guard Protection

6.1.3 Invalid Memory Access Protection

This is a feature that provides additional protection for detection of an invalid memory access.

Please note that once the “**IAWEN**” bit is set in the “IAWCTL” register, it cannot be disabled except for a Reset. Also, if the Watchdog is enabled in the Flash memory Option Bytes registers, then the invalid memory protection automatically enabled.

If an invalid memory access is detected, then an internal Reset is generated. The Reset source can be determined by examining the “RESF” register. The “IAWRF” bit will be set if the invalid memory access was the source of the Reset.

Format of Invalid Memory Access Detection Control Register (IAWCTL)

Address: F0078H Reset: 00H R/W

Symbol	7	6	5	4	3	2	1	0
IAWCTL	**IAWEN**	0	GRAM1	GRAM0	0	GPORT	GINT	GCSC

IAWEN	Control of illicit memory access detection
0	Disables the detection of illicit memory accesses.
1	Enables the detection of illicit memory accesses.

Figure 6-3 Invalid Memory Access Protection

6.1.4 I/O Port SFR Protection

This is a write protection feature that prohibits a write to the SFR registers. No error is generated if a write occurs, but the write operation does not change the state of the registers involved. Please note that the data port register (Pxx) cannot be protected.

The protection can be turned off, if a change is required for the SFR registers or for safety reasons the SFR settings are refreshed by the application.

The following I/O port SFR registers can be protected with this function.

PMxx, PUxx, PIMxx, POMxx, PMCAxx, PMCTxx, PMCExx, PFOEx,

PDIDISxx, CCDE, CCSm, PTDC, PFSEGx and ISCLCD

Pxx cannot be guarded.

The Port I/O SFR registers can be guarded by the "**GPORT**" bit as shown in Figure 6-4 below

Format of Invalid Memory Access Detection Control Register (IAWCTL)

Address: F0078H Reset: 00H R/W

Symbol	7	6	5	4	3	2	1	0
IAWCTL	IAWEN	0	GRAM1	GRAM0	0	**GPORT**	GINT	GCSC

GPORT	Protection of the port control registers
0	Disabled. Reading from and writing to the port control registers are allowed.
1	Enabled. Writing to the port control registers is not allowed. Reading from them is allowed.

Figure 6-4 I/O Port SFR Guard Protection

6.1.5 Interrupt SFR Protection

This is a write protection feature that prohibits a write to the Interrupt SFR registers. No error is generated if a write occurs to this area, but the write operation does not change the state of the registers involved. The protection can be turned off, if a change is required for the SFR registers or for safety reasons the SFR settings are refreshed by the application.

The following interrupt registers can be protected with this function.

IFxx, MKxx, PRxx, EGPx and EGNx

The interrupt SFR registers can be guarded by the "**GINT**" bit as shown in Figure 6-5 below.

Format of Invalid Memory Access Detection Control Register (IAWCTL)

Address: F0078H Reset: 00H R/W

Symbol	7	6	5	4	3	2	1	0
IAWCTL	IAWEN	0	GRAM1	GRAM0	0	GPORT	**GINT**	GCSC

GINT	Protection of the interrupt control registers
0	Disabled. Reading from and writing to the interrupt control registers are allowed.
1	Enabled. Writing to the interrupt control registers is not allowed. Reading from them is allowed.

Figure 6-5 Guarding of Interrupt SFRs

6.1.6 Control Register Protection

This is a write protection feature that prohibits a write to the control registers. No error is generated if a write occurs to this area, but the write operation does not change the state of the registers involved. The protection can be turned off, if a change is required for the SFR registers or for safety reasons the SFR settings are refreshed by the application.

The following control registers can be protected with this function.

CMC, CSC, OSTS, CKC, PERx, OSMC, LVIM, LVIS, RPECTL, CKSEL,

PRRx, MOCODIV, WKUPMD, PSMCR, MODRV and SOMRG

The interrupt SFR registers can be guarded by the “**GCSC**” bit as shown in Figure 6-6 below.

Format of Invalid Memory Access Detection Control Register (IAWCTL)

Address: F0078H Reset: 00H R/W

Symbol	7	6	5	4	3	2	1	0
IAWCTL	IAWEN	0	GRAM1	GRAM0	0	GPORT	GINT	**GCSC**

GCSC	Protection of the clock, voltage detector, and RAM parity error detection control registers
0	Disabled. Reading from and writing to the clock, voltage detector, and RAM parity error detection control registers are allowed.
1	Enabled. Writing to the clock, voltage detector, and RAM parity error detection control registers is not allowed. Reading from them is allowed.

Figure 6-6 Invalid Memory Access Protection

7. Related Application Note

The application note related to this application note is listed below for reference.

- RL78 Family VDE Certified IEC60730/60335 Self Test Library APPLICATION NOTE（R01AN0749E）

Home page and Support Contact

Renesas Electronics Home Page

- https://www.renesas.com/

Contact for inquiries

- https://www.renesas.com/contact/

8. VDE certification status

Table 8-1 show the VDE certification status of each module (assembler file) constituting a library.

Table 8-1 VDE certification status of each module

Module	Ver.	**VDE certification status**
stl_RL78_RegisterTest.asm	3.00	Valid (code part is the same as VDE certified module)
stl_RL78_RegisterTest_psw.asm	3.00	
stl_RL78_RegisterTest_stack.asm	3.00	
stl_RL78_RegisterTest_cs.asm	3.00	
stl_RL78_RegisterTest_es.asm	3.00	
stl_RL78_RegisterTest_pc.asm	3.01	
stl_RL78_peripheral_crc.asm	3.00	
stl_RL78_march_c.asm	3.00	
stl_RL78_march_x.asm	3.01	
stl_RL78_march_c_initial.asm	3.01	
stl_RL78_march_x_initial.asm	3.01	
stl_RL78_hw_clocktest.asm	3.01	
stl_RL78_hw_clocktestElc.asm	3.00	
stl_adc.c	3.01	
stl_RL78_GpioTest.asm	3.00	
stl_RL78_RamTest_Stacks_c.asm	3.03	
stl_RL78_RamTest_Stacks_x.asm	3.03	

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	30.Sep.2025	-	First edition issued

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 be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. 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.
6. 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.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. 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.
9. 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.
10. 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.
11. 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.
12. 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.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. 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.5.0-1 October 2020)

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/.