RL78 Family

IEC 60730/60335 Self Test Library for RL78 MCU (Class-C)

R01AN7508EJ0101
Rev.1.01
Oct.20.25

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.

This Application Note provides guidelines on how to use flexible sample software routines to assist with compliance with IEC60730 class C 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

RL78/G23 MCU
RL78/G14 MCU
RL78/F24 MCU

Contents

1. Overview of Self-test Library....4

2. Self-test Library Functions....6
2.1 Instruction Decoding Test....6
2.1.1 CPU Instruction Test – Software API....8
2.2 CPU Register Test....10
2.2.1 CPU Register Test – Software API....12
2.3 Invariable Memory Test – Flash ROM....18
2.3.1 CRC 16-CCITT algorithm....18
2.3.2 Hardware CRC – Software API....19
2.4 Variable Memory Test – SRAM....22
2.4.1 Algorithm....22
2.4.2 Variable Memory Test – Software API....23
2.5 System Clock Test....26
2.5.1 Hardware Measurement....26
2.6 Watchdog....30
2.6.1 Monitoring by built-in WDT....30
2.7 MCU Anomaly Detection....31

3. Example Usage....32
3.1 CPU....33
3.1.1 Power-on Tests....33
3.1.2 Periodic....33
3.2 Flash ROM....34
3.2.1 Power-on Tests....34
3.2.2 Periodic....34
3.3 RAM....35
3.3.1 Power-on Tests....35
3.3.2 Periodic....35
3.4 System Clock....36
3.4.1 Power-on Tests....36
3.4.2 Periodic....36

4. Development Environment....37
4.1 CS+ Settings....38
4.1.1 Common Options....38
4.1.2 Link Options....38
4.1.3 Hex Output Options....39
4.1.4 Download file for Debug Tool Configuration....40
4.1.5 Code Generation (Design Tool)....40
4.2 e^{2}stuido Settings....43

4.2.1 Complier Options 43
4.2.2 Assembler Options 44
4.2.3 Linker Options 44
4.2.4 Converter Options 45
4.2.5 Debug Configurations 46

5. Benchmark Test results 47

6. Related Application Note 48
Home page and Support Contact 48

Revision History 49

1. Overview of Self-test Library

The self-test library (STL) consists of self-test functions targeting instruction decode, CPU registers, internal memory, watchdog timer and system clock. As described below, the test harness provides an application program interface (API) for each module that monitors. Each function is used according to its purpose. The test harness uses the automatic generation function of the integrated development environment.

The self-test library functions are divided by module according to IEC60730 Class-C. In the test harness, each test function can be selected in turn and executed standalone.

The system hardware required at least two independent clock sources (e.g., crystal/ceramic oscillator and an independent operating oscillator or external input source). It is needed to provide an independent clock reference for monitoring the system clock. RL78 can use the High speed and Low speed internal oscillators which are independent of each other.

The self-test library is intended for implementation in a safety MCU. By integrating the safety section MCU into the system, risk reduction can be achieved from failures and other anomalies.

RL78 self-test library includes the following self-test functions:

- Instruction decoding

  Verifies that combinations of all addressing mode work correctly for all instructions of RL78 MCU

  Refer to “IEC 60730-1:2013+A1:2015+A2:2020 Annex H - H2.18.5 equivalence class test.”

- CPU register

  Test the following CPU registers.

  All CPU general-purpose registers in all 4 register banks, Stack Pointer (SP), Program Status Word (PSW), and extended register (CS and ES)

  The internal data path is verified during the normal operation test of the above registers

  Refer to “IEC 60730-1:2013+A1:2015+A2:2020 Annex H - Table H.11.12.7 1.CPU”

  Note: ABRAHAM algorithm is used for memory areas mapped as general-purpose registers.

- Invariable memory

  Test internal Flash memory of the MCU.

  Refer to “IEC 60730-1:2013+A1:2015+A2:2020 Annex H - H2.19.4.1 CRC - Single Word”

- Variable memory

  Test internal SRAM of the MCU

  Refer to “IEC 60730-1:2013+A1:2015+A2:2020 Annex H - H.2.19.1 Abraham test”

- System clock

  Test using TAU’s input capture feature against the system clock (this test requires an internal or external independent reference clock).

  Refer to “IEC Reference - IEC 60730-1:2013+A1:2015+A2:2020 Annex H - H2.18.10.1 Frequency monitoring”

- CPU/ Program Counter (PC)

  To confirm that the program is executing the sequence within the specified time, it is confirmed using the built-in watchdog timer that operates with a clock independent of the CPU. A process is implemented in the test harness to monitor whether the program is executing the expected sequence order.

  Refer to “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 Instruction Decoding Test

This test verifies that all instructions of RL78/G23 core in all addressing mode perform correctly. There are 2 types of execution: one with function calls from test harness process, and one without function calls from test harness process. The test startup without function calls from test harness process verifies all instructions before initializing “C” environment with a modified “startup.asm” module. If any anomaly is detected, stl_RL78_InstructionTest_Fail is called.

The target addressing mode and instructions are as follows.

For detail of addressing mode instructions, please refer to “RL78 Family User’s Manual: Software” (R01US0015).

(1) Instruction Address Addressing

There are 4 types of instruction address addressing as follows:

- Relative addressing
- Immediate addressing
- Table indirect addressing
- Register direct addressing

(2) Addressing for Processing Data address

There are 9 types of addressing for processing data address as follows:

- Implied addressing
- Register addressing
- Direct addressing
- Short direct addressing
- SFR addressing
- Register indirect addressing
- Based addressing
- Based indexed addressing
- Stack addressing

(3) RL78/G23 Instructions

RL78-S3 core has 81 types of instructions as follows:

【8-bit Data Transfer Instructions】

MOV XCH ONEB CLRB MOVS

【16-bit Data Transfer Instructions】

MOVW XCHGW ONEW CLRW

【8-bit Operation Instructions】

ADD ADDC SUB SUBC AND OR XOR CMP CMP0 CMPS

【16-bit Operation Instructions】

ADDW SUBW CMPW

【Multiply/Divide/Multiply & Accumulate Instructions】

MULU MULUH MULH DIVHU DIVWU MACHU MACH

【Increment/Decrement Instructions】

INC DEC INCW DECW

【Shift Instructions】

SHR SHRW SHL SHLW SAR SARW

【Rotate Instructions】

ROR ROL RORC ROLC ROLWC

【Bit Manipulation Instructions】

MOV1 AND1 OR1 XOR1 SET1 CLR1 NOT1

【Call Return Instructions】

CALL CALLT BRK RET RETI RETB

【Stack Manipulation Instructions】

PUSH POP

MOVW SP,src

MOVW AX,SP

ADDW SP,#Byte

SUBW SP,#byte

【Unconditional Branch Instructions】

BR

【Conditional Branch Instructions】

BC BNC BZ BNZ BH BNH BT BF BTCLR

【Conditional Skip Instructions】

SKC SKNC SKZ SKNZ SKH SKNH

【CPU Control Instructions】

SEL RBn NOP EI DI HALT STOP

Note: BRK, RETB, RETI, HALT, STOP are excluded from the instruction test.

2.1.1 **CPU Instruction Test – Software API**

Table 2-1 Source File: Periodic CPU instruction test

STL File name	Header File
stl_RL78_InstructionTest.asm	stl.h
Test Harness File Name	Header File
main.c	

Syntax	
unsigned char stl_RL78_InstructionTest (void)	
Description	
Test RL78 instructions other than the ones listed below EI DI The instructions listed above are tested in initial processing. The call function should not generate an interrupt during the test. Also, the test should start with register bank0 selected Test harness control file (main.c) calls stl_RL78_InstructionTest_Fail when an error is detected.	
Input Parameters	
None	N/A
Output Parameters	
None	N/A
Return Values	
unsigned char	Test result 0 = Test passed 1 = Test or parameter check failed

Table 2-2 Source File: Initial CPU instruction test

STL File	Header File
stl_RL78_InstructionTest.asm	stl_RL78_InstructionTest.inc
Test Harness File Names	Header File
startup.asm	

Syntax	
stl_RL78_InitialInstructionTest	
Description	
Test all RL78 instructions This module is executed before the application system is initialized. No function calls are used. When it finishes normally, jump to stl_RL78_InstructionTest_Pass When an error is detected, jump to stl_RL78_InstructionTest_Fail	
Input Parameters	
None	N/A
Output Parameters	
None	N/A
Return Vales	
None	N/A

2.2 CPU Register Test

This chapter describes each routine of the CPU register test. The test harness control file “main.c” contains API samples written in C language for each C register test.

These modules test the basic operation of CPU. Each API function informs the test result by return values.

The target CPU registers are as follows:

- General-Purpose Register: AX、HL、DE、BC of Register bank 0~3

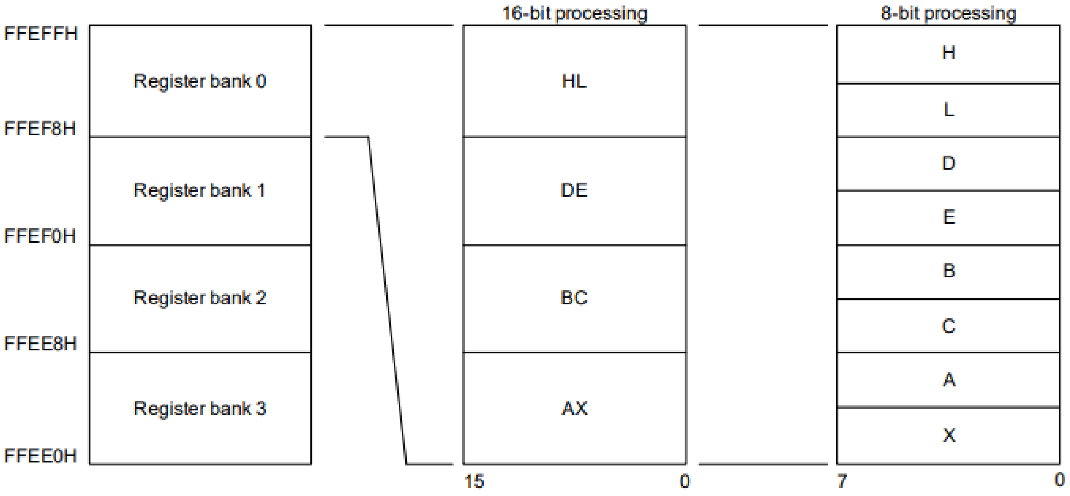


Figure 2-1 Configuration of General-Purpose Registers

- Stack Pointer (SP)

Symbol	15	14	13	12	11	10	9	8
SP	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8

	7	6	5	4	3	2	1	0
	SP7	SP6	SP5	SP4	SP3	SP2	SP1	0

Figure 2-2 Format of Stack Pointer

- Program Status Word (PSW)

Symbol	7							0
PSW	IE	Z	RBS1	AC	RBS0	ISP1	ISP0	CY

Figure 2-3 Format of Program Status Word

- CS register

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

Figure 2-4 Configuration of ES

- ES register

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

Figure 2-5 Configuration of ES

- Program Counter (PC)

Symbol	19 … 0
PC	

Figure 2-6 Format of Program Counter

2.2.1 CPU Register Test – Software API

Table 2-3 Source file: CPU general-purpose register test

STL File Name	Header File
stl_RL78_RegisterTest.asm	stl.h
Test Harness File Name	Header File
main.c	

Syntax	
unsigned char stl_RL78_RegisterTest(unsigned char Bank)	
Description	
Test RL78 general-purpose registers. Register AX, HL, DE, BC in specified register bank (Bank 0, 1, 2, 3) ABRAHAM algorithm is performed on memory areas mapped as registers. It is the calling function’s responsibility to ensure no interrupts occur during this test. In addition, Register Bank 0 must be selected when this test starts. The original register contents are restored on completion of the test. The function RegisterTest_Failure is called by the test harness control file (main.c) when an error is detected.	
Input Parameters	
unsigned char Bank	Target register bank (0~3)
Output Parameters	
None	N/A
Return Values	
unsigned char	Test result 0 = Test passed 1 = Test or parameter check failed

Table 2-4 Source file: CPU register test – PSW

STL File Name	Header File
stl_RL78_registertest_psw.asm	stl.h
Anomality Monitoring Process File Name	Header File
r_main.c	

Syntax	
unsigned char stl_RL78_registertest_psw(void)	
Description	
Test 8-bit Program Status Word (PSW) register The following tests are performed: 1. Write h'55 to the register 2. Read out the register and check that it is equal to the written value 3. Write h'AA to the register 4. Read out the register and check that it is equal to the written value It is the calling function’s responsibility to ensure no interrupts occur during this test. The original register content is restored on completion of the test 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	
unsigned char	Test result 0 = Test passed 1 = Test or parameter check failed.

Table 2-5 Source file: CPU register test - SP

STL File Name	Header File
stl_RL78_registertest_stack.asm	stl.h
Anomality Monitoring Process File Name	Header File
r_main.c	

Syntax	
unsigned char stl_RL78_registertest_stack(void)	
Description	
Test 16-bit Stack pointer (SP) register The following tests are performed 1. Write h'5555 to the register 2. Read out the register and check that it is equal to h'5554 3. Write h'AAAA to the register 4. Read out the register and check that it is equal to the written value It is the calling function's responsibility to ensure no interrupts occur during this test. The original register content is restored on completion of the test 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	
unsigned char	Test Result 0 = Test passed。 1 = Test or parameter check failed

Table 2-6 Source file: CPU register test – CS

STL File Name	Header File
stl_RL78_registertest_cs.asm	stl.h
Anomality Monitoring Process File Name	Header File
r_main.c	

Syntax	
unsigned char stl_RL78_registertest_cs(void)	
Description	
Test the 8-bit CS register The following tests are performed. 1. Write h'05 to the register 2. Read out the register and check that it is equal to the written value 3. Write h'0A to the register 4. Read out the register and check that it is equal to the written value Please note that the top 4 bits are fixed to '0' It is the calling function's responsibility to ensure no interrupts occur during this test. The original register content is restored on completion of the test 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	
unsigned char	Test result 0 = Test passed 1 = Test or parameter check failed

Table 2-7 Source file: CPU register test – ES

STL File Name	Header File
stl_RL78_registertest_es.asm	stl.h
Anomality Monitoring Process File Name	Header File
r_main.c	

Syntax	
unsigned 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 2. Read out the register and check that it is equal to the written value 3. Write h'0A to the register 4. Read out the register and check that it is equal to the written value Please note that the top 4 bits are fixed to ‘0’ It is the calling function’s responsibility to ensure no interrupts occur during this test. The original register content is restored on completion of the test 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	
unsigned char	Test Result 0 = Test passed 1 = Test or parameter check failed

Table 2-8 Source file: CPU register test – PC

STL File Name	Header File
stl_RL78_Registertest_pc.asm	stl.h
Test Harness File Name	Header File
main.c	

Syntax	
unsigned char stl_RL78_RegisterTest_pc(void)	
Description	
The following tests are performed 1. Set the argument (A register) to 0x55 2. Call a function that returns the reversed result of the argument 3. Confirm that the return value is 0xAA 4. Confirm that the return value is equal to the return address. It is the calling function’s responsibility to ensure no interrupts occur during this test. 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	
unsigned char	Test result 0 = Test passed 1 = Test or parameter check failed

2.3 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 analyze 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 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. Any of the test functions can be repeated, thus allowing the option of a full CRC calculation to be made or a CRC calculation of a smaller segments suitable to the operation of the end application. For a full calculation (or first part of an iterative calculation), an initial value of h'0000 is used or the previous partial result is provided as the starting point for the next calculation stage.

The hardware module is "the general-purpose CRC function" embedded in RL78 device. The hardware module while using the same fundamental CRC algorithm uses a different data format (LSB first) for calculating the reference CRC value.

As a note for debugging, when a software break is set in the debugger, the instruction code at the specified address is temporarily rewritten to the instruction for the break since any of the test functions can be executed repeatedly. Therefore, CRC mismatch may occur.

2.3.1 CRC 16-CCITT algorithm

RL78 CRC module supports CRC16-CCITT.

Hardware algorithm

- CCITT 16 Polynomial = 0x1021 ($x^{16} + x^{12} + x^5 + 1$)
- Input Data width = 8bits
- LSB first (operation is performed with the bit order reversed on input, and the operation result is also output with the bit order reversed)
- Initial value = 0x0000 or 16 bit result of previous partial CRC calculation

2.3.2 Hardware CRC – Software API

Table 2-9 Source file: Hardware CRC calculation

STL File Name	Header File
stl_RL78_peripheral_crc.asm	stl.h
Test Harness File	Header File
main.c	

Syntax	
unsigned short stl_RL78_peripheral crc(unsigned short crc, CHECKSUM_CRC_TEST_AREA *p)	
Description	
This function calculates a CRC value over the address range supplied using the hardware CRC peripheral (general purpose CRC). The start address and calculation range (Length) are passed by the calling function via the structure detailed in the table below. The calculated result is returned. This can be either a partial result of full result depending upon the parameters provided. The harness file (main.c) compares the result of calculated CRC in the divided areas. Note: Set the same value as that set in CC-RL for the range subject to CRC check and the address where CRC value is saved in the invariable memory.	
Input Parameters	
unsigned short crc	Initial value for CRC calculation (0 is specified only for the first block, otherwise it is the previous result)
CHECKSUM_CRC_TEST_AREA *p	Pointer to structure that stores the start address and calculation range
Output Parameters	
None	N/A
Return Values	
unsigned short	16bit CRC value (Full or partial result)

Table 2-10 Source file: Hardware CRC parameter structure

Syntax	
static CHECK_CRC_TEST_AREA lv_CheckCrc;	
Description	
Structure declaration and instance providing the parameters to be passed to the hardware CRC module (stl_RL78_peripheral_crc.asm) by the calling function in main.c.	
Input Parameters	
unsigned long m_length;	Range (length = number of bytes) of target memory
unsigned long m_start_address	Start address for CRC calculation
Output parameters	
None	N/A
Return Values	
None	N/A

Definition of test harness CRC calculation target

```
typedef struct CRC_RANGE
{
  uint32_t Start;
  uint32_t End;
}CRC_RANGE;
```

The ROM test calculates the CRC value in units of 32 Kbytes and checks for a match with the CRC saved in a specific area.

Note: On-chip debugger occupies 512bytes from the last address in ROM. Therefore, the CRC storage address is the last address of ROM – 512 – (Number of blocks*2) at the beginning.

```
#define CRC_RANGE_NUM (sizeof(CRC_Ranges)/sizeof(CRC_RANGE))
const CRC_RANGE CRC_Ranges[] =
{
  {0x00000,0x07FFF},                    /*  32K */
  {0x08000,0x0FFFF},                    /*  64K */
  {0x10000,0x17FFF},                    /*  96K ,0x17FFF - 512 - (3 * 2) */
  {0x18000,0x1FFFF},                    /* 128K ,0x1FFFF - 512 - (4 * 2) */
  {0x20000,0x27FFF},                    /* 160K */
  {0x28000,0x2FFFF},                    /* 192K ,0x2FFFF - 512 - (6 * 2) */
  {0x30000,0x37FFF},                    /* 224K */
  {0x38000,0x3FFFF},                    /* 256K ,0x3FFFF - 512 - (8 * 2) */
  {0x40000,0x47FFF},                    /* 288K */
  {0x48000,0x4FFFF},                    /* 320K */
  {0x58000,0x5FFFF},                    /* 384K ,0x5FFFF - 512 - (12 * 2)  */
  {0x60000,0x67FFF},                    /* 416K */
```

```
  {0x68000,0x6FFFF},                   /* 448K */
  {0x70000,0x77FFF},                   /* 480K */
  {0x78000,0x7FFFF},                   /* 512K ,0x7FFFF - 512 - (16 * 2) */
  {0x80000,0x87FFF},                   /* 544K */
  {0x88000,0x8FFFF},                   /* 576K */
  {0x90000,0x97FFF},                   /* 608K */
  {0x98000,0x9FFFF},                   /* 640K */
  {0xA0000,0xA7FFF},                   /* 672K */
  {0xA8000,0xAFFFF},                   /* 704K */
  {0xB0000,0xB7FFF},                   /* 736K */
  {0xB8000,0xBFFFF - 512 - (24 * 2)}   /* 768K */
};
```

Please change with the target MCU.

Definition of CRC calculation result storage area

It is defined in stl.h.

```
#define DEF_ROM_CRC      (0xBFDD0)
```

Please change with the target MCU.

2.4 Variable Memory Test – SRAM

ABRAHAM test is a method of RAM test that meets IEC 60730-1:2013+A1:2015+A2:2020 Annex H – H2.19.1.

The algorithm itself is destructive and does not save the current RAM values. Therefore, RAM contents must be saved if tests are performed after initialization of the application system or during running. The additional test module (stl_RL78_InitialRamTest) is designed to be executed before initializing the system, so that the whole memory area can be tested before starting the main application.

The RAM area to be tested can't be used for any other purpose during testing. This 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 process is finished.

The next chapter describes the ABRAHAM test.

2.4.1 Algorithm

(1) ABRAHAM

ABRAHAM algorithm consists of 10 elements that perform 30 different processes in total. It detects the following faults:

1. Stuck At Faults (SAF)

- The logic value of a single cell or contiguous cells is always 0 or 1

2. Transition Faults (TF)

- A single cell or contiguous cells does not transit from 0→1 or 1→0

3. Coupling Faults (CF)

- The state or transition of a cell value causes the value of other cell to change

4. Address Decoder Faults (AF)

- Failure affects address decoding
- Unable to access a certain address cell
- Unable to access a certain cell
- Unable to access multiple cells simultaneously with a certain address
- A certain cell is accessed from multiple addresses

⇔ (w0)	Initialize
↓ (r0, w1) ↑ (r1)	Sequence 1
↓ (r1, w0) ↑ (r0)	Sequence 2
↑ (r0, w1) ↓ (r1)	Sequence 3
↑ (r1, w0) ↓ (r0)	Sequence 4
↓ (r0, w1, w0) ↑ (r0)	Sequence 5
↑ (r0, w1, w0) ↑ (r0)	Sequence 6
⇔ (w1)	Reset
↑ (r1, w0, w1) ↑ (r1)	Sequence 7
↓ (r1, w0, w1) ↑ (r1)	Sequence 8

⇔ : Perform in ascending or descending order of address
↓ : Perform in a descending order of the address
↑ : Perform in an ascending order of the address
w0 : write 0 to the cell
w1 : write 1 to the cell
r0 : Read 0 from the cell as expected
r1 : Read 1 from the cell as expected

2.4.2 Variable Memory Test – Software API

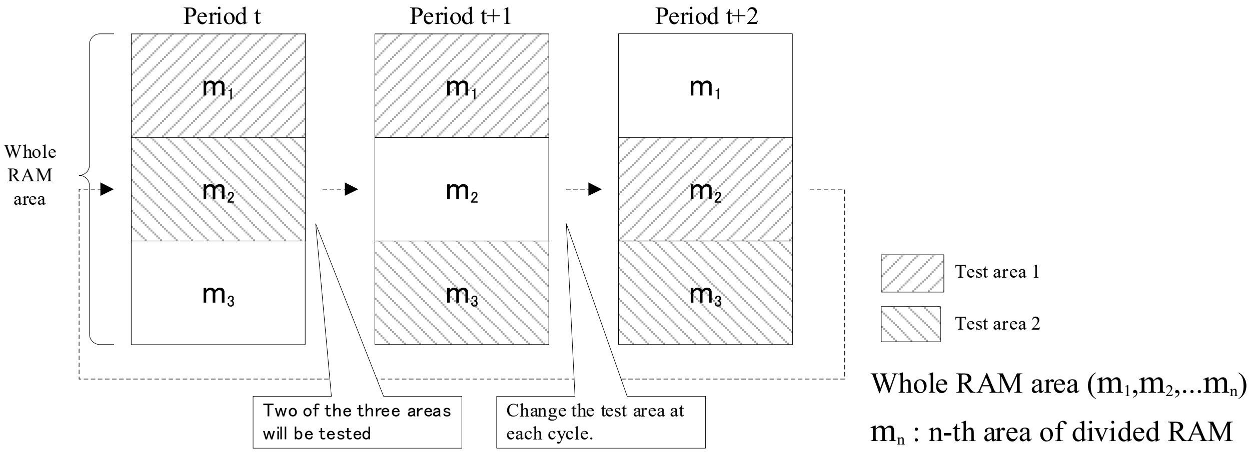
2.4.2.1 Time Division ABRAHAM

Execute after initialization of the application system. Uses C stack resources for execution with normal function calls from the test harness. It is possible to test part or all of RAM area, but the area to be tested must be buffered because it is destructive. Therefore, it is not recommended to test the entire RAM area in a single run. Also, be careful that the test itself does not destroy the RAM area used as the stack area. The time division ABRAHAM executes ABRAHAM by treating the divided areas.

⇔[x,y] (w0)	Initialize	⇔[x,y]:	Perform in ascending or descending order
↓ [x,y](r0, w1) ↑ (r1)	Sequence 1		of address in the range of x and y
↓ [x,y](r1, w0) ↑ (r0)	Sequence 2	↓[x,y] :	Perform in descending order of address
↑ [x,y](r0, w1) ↓ (r1)	Sequence 3		in the range of x and y
↑[x,y] (r1, w0) ↓ (r0)	Sequence 4	↑[x,y]:	Perform in ascending order of address
↓ [x,y] (r0, w1, w0) ↑ (r0)	Sequence 5		in the range of x and y
↑ [x,y] (r0, w1, w0) ↑ (r0)	Sequence 6	w0:	write 0 to the cell
⇔ [x,y] (w1)	Reset	w1:	write 1 to the cell
↑ [x,y] (r1, w0, w1) ↑ (r1)	Sequence 7	r0:	Read 0 from the cell as expected
↓ [x,y] (r1, w0, w1) ↑ (r1)	Sequence 8	r1:	Read 1 from the cell as expected

3-partition case is shown in Figure 2-7. [x,y] is [m1,m2], [m1,m3], [m2,m3] respectively.

For example, if the memory is divided into 3 parts, testing the combination of m1 and m2 at period t, m1 and m3 at period t+1, m2 and m3 at period t+2, the results are equivalent to the test the entire memory at once.



Whole RAM area (m_1,m_2,...m_n)

m_n : n-th area of divided RAM

Figure 2-7 Overview of Time Division ABRAHAM

Table 2-11 Source file: Variable memory test

STL File Name	Header File
stl_RL78_Ram.asm	stl.h
Test Harness File	Header File
main.c	

Syntax	
unsigned char stl_RL78_RamTest (unsigned char *pRam1, unsigned char *pRam2, unsigned short Size)	
Description	
Test the address range of RAM specified by the calling function using the time division ABRAHAM algorithm and return the result (pass/fail). This module should be executed after initialization of the application system. The calling function must always start with register bank 0 selected. The contents of the area under test must be saved beforehand. The test is executed destructively. Use register bank 1 as the work area.	
Input Parameters	
unsigned char *pRam1	Pointer to the first address of RAM1 to be tested
unsigned char *pRam2	Pointer to the first address of RAM2 to be tested
unsigned short Size	RAM range to be tested (number of bytes)
Output Parameters	
None	N/A
Return values	
unsigned char	Test result 0 = Test passed 1 = Test or parameter check failed

2.4.2.2 Initial ABRAHAM

Initial ABRAHAM test is performed before the application system is initialized. It does not use function calls from the test harness for execution. The test is started by a jump from the modified "startup.asm" module, and a return to the "startup.as" module is also done by a jump. The test results are stored in the 8-bit accumulator(A). Thus, the entire area of RAM can be tested before booting the system and initializing the "C" environment.

Table 2-12 Source file: Initial ABRAHAM

STL File Name	Header File
stl_RL78_InitialmRam.asm	None
Test Harness File	Header File
startup.asm	None

Syntax	
stl_RL78_InitialRamTest	
Description	
Test the address range of RAM specified by the calling function using the time division ABRAHAM algorithm and return the result (pass/fail). This module should be executed after initialization of the application system. No function call is used. The test result if done through the function 'stl_RL78_InitialRamTestResult'. Note: the function 'stl_RL78_InitialRamTestResult' is in the module main.c.	
Input Parameters	
CPU register AX	16-bit register that stores the first address of the target RAM
CPU register BC	16-bit register that stores the target RAM range (number of bytes)
Output Parameters	
None	N/A
Return Values	
CPU register A	Test result 0 = Test passed 1 = Test or parameter check failed

2.5 System Clock Test

A self test module provided for RL78 self test library in order to be able to test the internal system clock (CPU and Peripheral clocks). These modules can be used by the application to detect the correct operation and deviation in the main system clock during operation of the application. Please note that if the internal low speed oscillator is used for measurement, the accuracy of the system clock measurement will be reduced due to the greater tolerance of the internal low speed oscillator. Therefore, only the relative operation of the system clock can be obtained, which should still be sufficient to establish that the system clock is operating correctly and within acceptable limits.

The principle behind both measurement approaches is that the system can detect if the operating frequency of the main clock deviates from a predefined range during runtime. The accuracy of the measurement obviously depends on the accuracy of the reference clock source. For example, an external signal input or 32KHz crystal can provide a more accurate measurement for the system clock than the internal low speed oscillator. However, it requires extra components.

A “Pass / Fail” status of the test is returned. Also implemented is a “No Reference Clock” detection scheme which returns a different status value to the normal test, to identify the appropriate fault state. The module compares the measure (captured) time is within a reference window (upper and lower limit values) using the user defined reference values set in ‘stl_RL78_hw_clocktest.inc’ header file. The header file defines the reference values for hardware measurement and input test port pins.

2.5.1 Hardware Measurement

All current RL78 devices include an option in the Timer Array Unit (TAU) channel 5 that provides additional input capture sources that are designed to be able to test the system clock operation. The extra capture inputs are selected as part of the “safety” register (TIS0) and include the following:

- Internal Low-speed oscillator (fiL)
- External 32KHz oscillator (Sub Clock) (fsub)
- External signal input (TIO5)

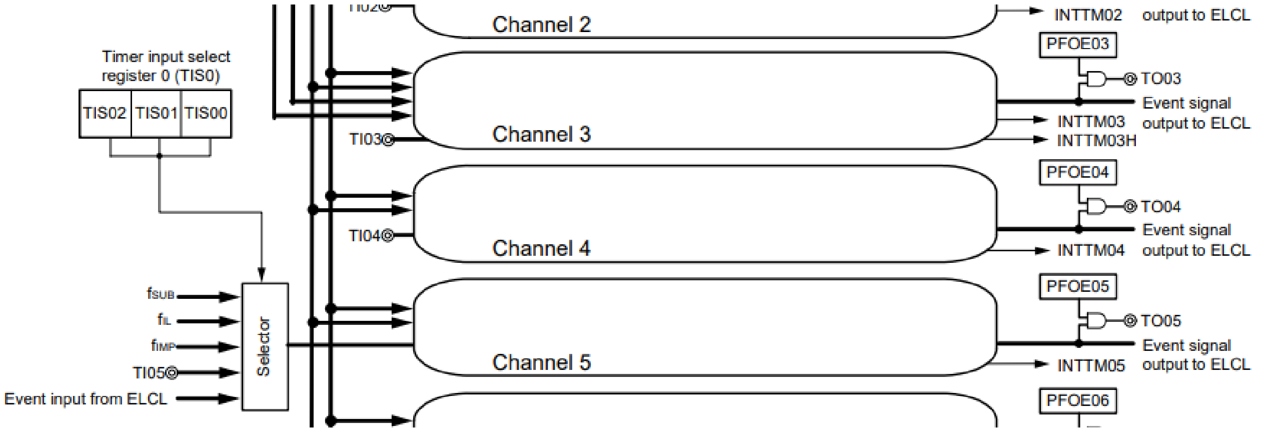


Figure 2-8 Timer Array Unit Channel 5 configuration

Note: In RL78/G14, F24, Channel 1 is the measurement channel.

The principle behind the hardware measurement is based on the input capture measurement of the reference clock in TAU channel 5. As this is a hardware capture measurement the time captured is the “period” of the reference clock as that of the system clock.

The measurement sequence is.

- Synchronize 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 anomaly monitoring process provides a sample based on the following settings.

System clock = 32 MHz

Reference clock = 32.768 kHz

Therefore, the calculation is 32000000/32768 = 976

Note: The internal low-speed oscillator of RL78/G14, F24 is 15kHz.

The capture value should be set to the allowed fluctuation range relative to the upper and lower limits of the reference values.

【Timer setting (setting of automatic generation function)】

Channel 5: Input pulse interval measurement

Timer input; fIL

Pin input valid range: Rising edge.

Interrupt: not generated

Note: RL78/G14, F24 use Channel 1

Table 2-13 Source File: Hardware Clock test

STL File Name	Header File
stl_RL78_hw_clocktest.asm	stl_RL78_hw_clocktest.inc stl.h
Test Harness File	Header File
main.c	

Syntax	
void stl_RL78_Init_hw_clocktest (unsigned char Select)	
Description	
Start capturing the system clock using hardware measurements (TAU channel 5)	
Input Parameters	
Select	Input of TAU Channel 5 0: Input signal of timer input pin (TI05) 5: Subsystem clock (fsub) Other: Low-speed on-chip oscillator clock (fIL)
Output Parameters	
None	N/A
Return Values	
None	N/A

Syntax	
unsigned char stl_RL78_hw_clocktest(void)	
Description	
This function tests the system clock using the hardware measurement (TAU channel 5) feature. The measured result (capture value) is compared against the upper and lower limit values defined in the clock test header file (stl_RL78_hw_clocktest.inc) and the result status (Pass / Fail / No reference clock) is returned to the calling function.	
Input Parameters	
hwMAXTIME	Upper time limit compare value (defined in stl_RL78_hw_clocktest.inc)
hwMINTIME	Lower time limit compare value (Defined in stl_RL78_hw_clocktest.inc)
CAPTURE_interrupt_FLAG	Timer channel Capture Interrupt Flag (stl_RL78_hw_clocktest.inc)
Output Parameters	
None	N/A
Return Values	
unsigned char	Test result 0 = Test passed. 1 = Test measurement failed (Outside the reference window) 2 = Test measurement failed (No reference clock detected)

2.6 Watchdog

A built-in watchdog timer monitors whether the program is operating as expected.

【Watchdog setting (setting of automatic generation function)】

Waterdog timer: enable.

Watchdog timer operation in HALT/STOP mode: continues.

Overflow Time: 125ms(2^12/fIL)

Window Open Period: 50%

Interval interrupt is generated when 75% of the overflow time + 1/4 fIL is reached: not used.

2.6.1 Monitoring by built-in WDT

Table 2-14 Source file: built-in WDT clear

STL File Name	Header File
Config_WDT.c	Config_WDT.h
Test Harness File	Header File
main.c	

Syntax	
void R_Config_WDT_Restart (void)	
Description	
Refresh the built-in WDT	
Input Parameters	
None	N/A
Output Parameters	
None	N/A
Return Values	
Noe	N/A

2.7 MCU Anomaly Detection

RL78/G23 has SFR to check the reset factor. The self-test library can be configured to analyze the reset factor and call the following function when it is not a power-up reset.

Factor	Option settings (stl_RL78_TestConfig.inc)	Calling functions
Header File	Illegal_MemoryAccess_enabled	Illegal_MemoryAccess
RAM parity error	RAM_Parity_Failure_enabled	RAM_Parity_Failure
Watchdog timer over	Watchdog_Test_enabled	Watchdog_Test_Failure
Execution of illegal orders	Illegal_InstructionExecution_enabled	Illegal_InstructionExecution
Voltage abnormal	Voltage_Test_Reset_enabled	Voltage_Test_Failure_Reset

Note: RL78/F24 has no RAM parity error.

3. Example Usage

In addition to the actual test software source files, the CS+/e^{2}studio 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 are 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 instructions test
- Initial RAM test using ABRAHAM algorithm
- All register test
- Flash memory CRC test

The clock test may be run at a later time depending on the initial clock speed if the clock is to establish that the maximum clock speed is to be measured.

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

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

- Register test
  These are dependent upon the application timing.

- Periodic instruction test
  These are dependent upon the application timing.

- Flash memory test
  These modules are designed to be able to accumulate a CRC result over a number of 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

If a fault is detected by any of the CPU tests, then this is very serious. The aim of this test should be to get to a safe operating point, where software execution is not relied upon, as soon as possible.

3.1.1 Power-on Tests

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

3.1.2 Periodic

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 is tested by calculating a CRC value over a certain range of the Flash memory 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 CS+ /e^{2}Studio tool chain can be used to calculate and add a CRC value and place at a location specified by the user. CS+ /e^{2}Studio grants three types of CRC: “general-purpose CRC”, “high-speed CRC (CCR-16-CCITT)”, and “high-speed CRC (SENT)”. Hardware CRC calculation provides in this library (function “stl_RL78_peripheral_crc”) corresponds to the “general-purpose CRC”. See Figure 4-3 CS+/ e^{2}Studio Hex Output Options for how to incorporate the CRC value in CC-RL

3.2.1 Power-on Tests

All the ROM memory used must be tested at power up. Both hardware and software CRC modules are capable of calculating the CRC value over the whole memory range.

3.2.2 Periodic

It is suggested that the periodic testing of Flash memory is done in stages, depending on the time available to the application. The application will need to save the partially calculated result if using the software module. This value can then be set as starting point for the next stage of the CRC calculation.

When using the hardware peripheral unit, the partial CRC result value could be left in the result register of the hardware CRC peripheral unit, but it is advised to save this value and compare it before starting the next part of the calculation.

In this way all of the Flash memory can be verified in time slots convenient to the application.

3.3 RAM

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.
- The stack area cannot be tested after initializing the system unless its contents are relocated to another area and the stack pointer is changed accordingly. Also, no interrupt processing is allowed during the operation.

3.3.1 Power-on Tests

All instructions test of MCU is performed at Power-on or reset. If the test fails, it jumps to stl_RL78_InstructionTest_Fail without calling the main function. It is recommended to use initial RAM test modules after all instructions test is done. These modules are designed specifically for testing all RAM areas at Power-on or reset. They are also suitable to be executed before initializing the system and C stack, since they do not require function calls but destroy the RAM contents. In this library, the initial RAM test is implemented in the assembler file startup.asm.

3.3.2 Periodic

Periodic testing of the Ram memory is usually done in small stages, depending on the time available to the application and the available space necessary to buffer the system Ram contents during testing. Each stage provides a pass / fail status over the range specified, in this way all of the Ram memory can be verified time slots convenient to the application.

3.4 System Clock

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

3.4.1 Power-on Tests

The system clock should be verified at power on or reset. It may be necessary to test the clock once the system has been initialized and the full system clock frequency has been set and stabilized.

3.4.2 Periodic

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 in order to increase the accuracy of the clock measurement.

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

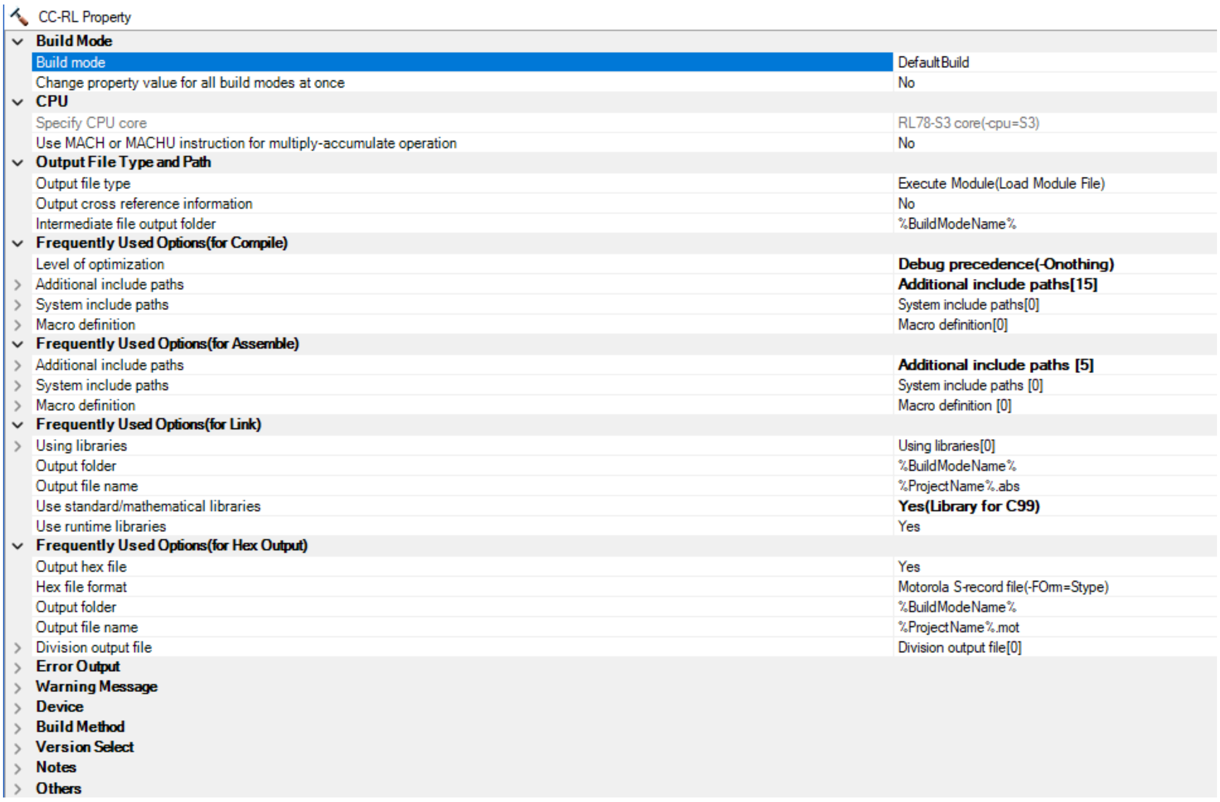
Note: The reference clock of RL78/G14, F24 is 15KHz.

4. Development Environment

• E2-Lite	On-chip debugging emulator
• RL78/G23 Fast Prototyping Board	RL78/G23（128 pin LFQFP）
• Tool chain	CS+ Version 8.1.0.00 CC-RL Version1.13.0 e2Studio 2024-01.1 CC-RL Version1.13.0
• MCU	R7F100GSN2DFB
• Internal clock	32 MHz High speed on-chip Oscillator System clock = 32 MHz
• Low speed clock	32 kHz Low speed on-chip oscillator

4.1 CS+ Settings

4.1.1 Common Options

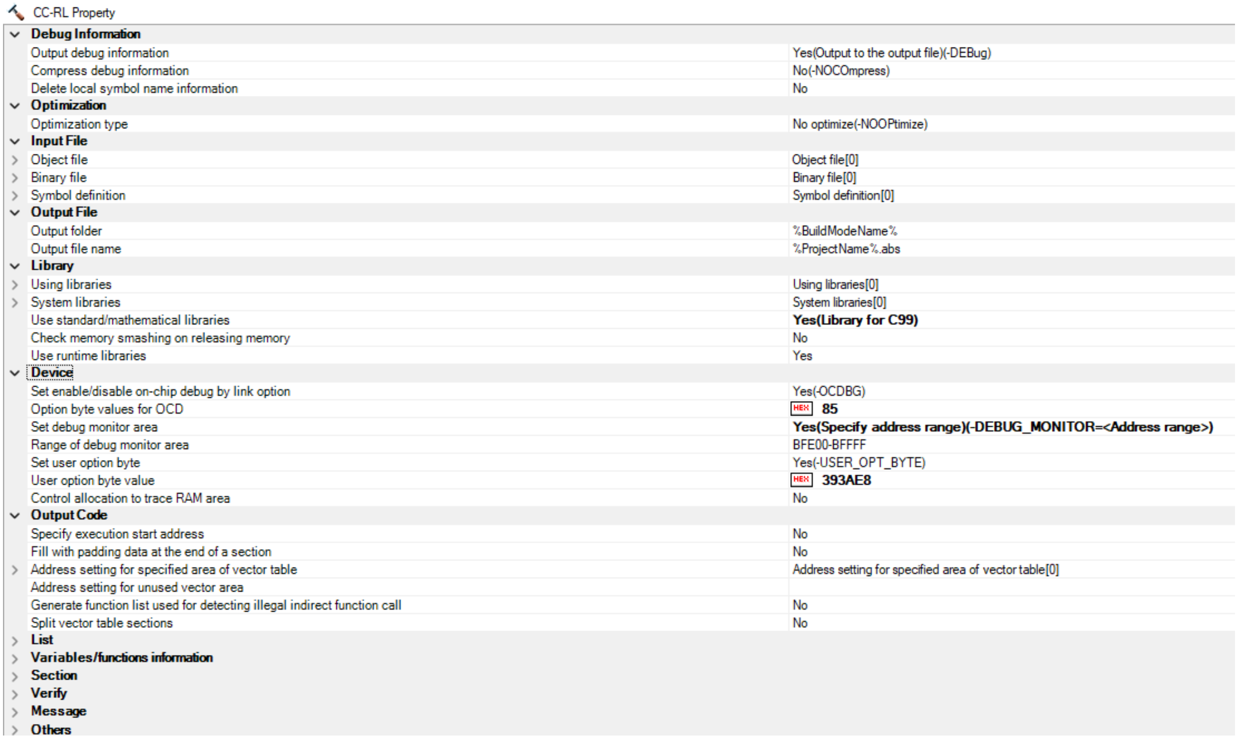


CC-RL Property

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[15]**
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	
Build Method	
Version Select	
Notes	
Others	

Figure 4-1 CS+ Common Options

4.1.2 Link Options



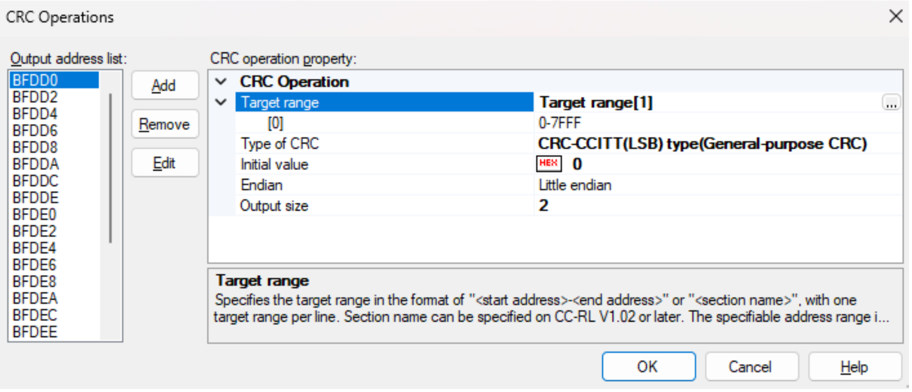
CC-RL Property

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	BFE00-BFFFF
Set user option byte	Yes(-USER_OPT_BYTE)
User option byte value	HEX **393AE8**
Control allocation to trace RAM area	No
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	

Figure 4-2 CS+ Link Options

4.1.3 Hex Output Options

CRC Operation	
CRC operations	**CRC operations[24]**
[00]	BFDD0=0-7FFF/16-CCITT-LSB(0):LITTLE-2-0
[01]	BFDD2=8000-FFFF/16-CCITT-LSB(0):LITTLE-2-0
[02]	BFDD4=10000-17FFF/16-CCITT-LSB(0):LITTLE-2-0
[03]	BFDD6=18000-1FFFF/16-CCITT-LSB(0):LITTLE-2-0
[04]	BFDD8=20000-27FFF/16-CCITT-LSB(0):LITTLE-2-0
[05]	BFDDA=28000-2FFFF/16-CCITT-LSB(0):LITTLE-2-0
[06]	BFDDC=30000-37FFF/16-CCITT-LSB(0):LITTLE-2-0
[07]	BFDDE=38000-3FFFF/16-CCITT-LSB(0):LITTLE-2-0
[08]	BFDE0=40000-47FFF/16-CCITT-LSB(0):LITTLE-2-0
[09]	BFDE2=48000-4FFFF/16-CCITT-LSB(0):LITTLE-2-0
[10]	BFDE4=50000-57FFF/16-CCITT-LSB(0):LITTLE-2-0
[11]	BFDE6=58000-5FFFF/16-CCITT-LSB(0):LITTLE-2-0
[12]	BFDE8=60000-67FFF/16-CCITT-LSB(0):LITTLE-2-0
[13]	BFDEA=68000-6FFFF/16-CCITT-LSB(0):LITTLE-2-0
[14]	BFDEC=70000-77FFF/16-CCITT-LSB(0):LITTLE-2-0
[15]	BFDEE=78000-7FFFF/16-CCITT-LSB(0):LITTLE-2-0
[16]	BFDF0=80000-87FFF/16-CCITT-LSB(0):LITTLE-2-0
[17]	BFDF2=88000-8FFFF/16-CCITT-LSB(0):LITTLE-2-0
[18]	BFDF4=90000-97FFF/16-CCITT-LSB(0):LITTLE-2-0
[19]	BFDF6=98000-9FFFF/16-CCITT-LSB(0):LITTLE-2-0
[20]	BFDF8=A0000-A7FFF/16-CCITT-LSB(0):LITTLE-2-0
[21]	BFDFA=A8000-AFFFF/16-CCITT-LSB(0):LITTLE-2-0
[22]	BFDFC=B0000-B7FFF/16-CCITT-LSB(0):LITTLE-2-0
[23]	BFDFE=B8000-BFDCF/16-CCITT-LSB(0):LITTLE-2-0
Displays the result of CRC calculation and output address	**Yes(-VERBOSE=CRC)**



Output address list:

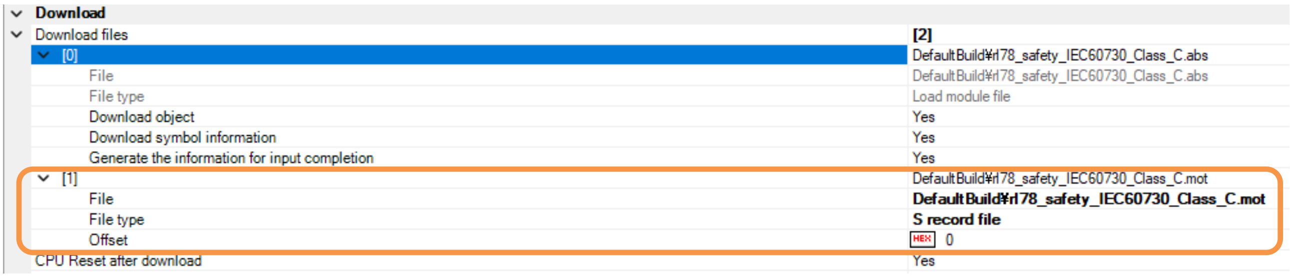
BFDD0
BFDD2
BFDD4
BFDD6
BFDD8
BFDDA
BFDDC
BFDDE
BFDE0
BFDE2
BFDE4
BFDE6
BFDE8
BFDEA
BFDEC
BFDEE

CRC Operation	
Target range	**Target range[1]**
[0]	0-7FFF
Type of CRC	**CRC-CCITT(LSB) type(General-purpose CRC)**
Initial value	HEX **0**
Endian	Little endian
Output size	**2**

Target range
Specifies the target range in the format of "<start address>-<end address>" or "<section name>", with one target range per line. Section name can be specified on CC-RL V1.02 or later. The specifiable address range i...

Figure 4-3 CS+ Hex Output Options

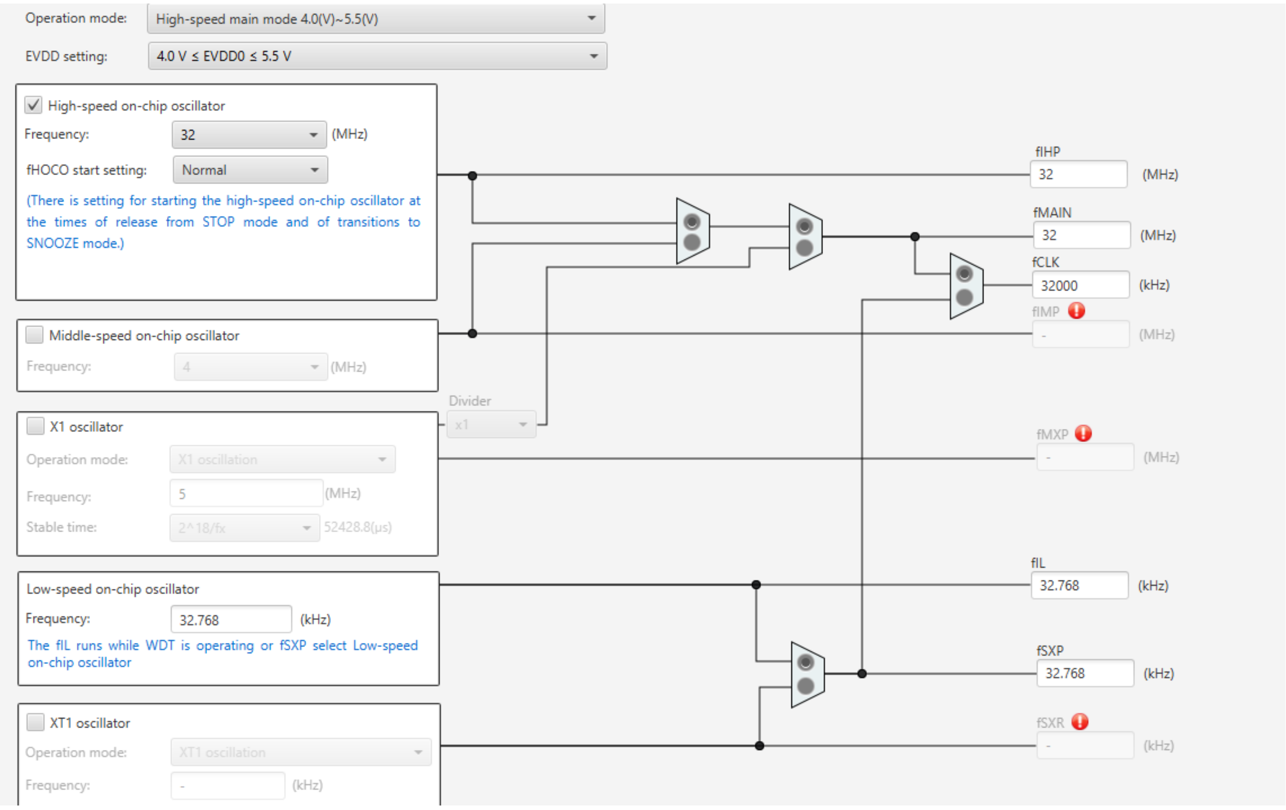
4.1.4 Download file for Debug Tool Configuration



Download	
Download files	**[2]**
[0]	DefaultBuild¥rl78_safety_IEC60730_Class_C.abs
File	DefaultBuild¥rl78_safety_IEC60730_Class_C.abs
File type	Load module file
Download object	Yes
Download symbol information	Yes
Generate the information for input completion	Yes
[1]	DefaultBuild¥rl78_safety_IEC60730_Class_C.mot
File	**DefaultBuild¥rl78_safety_IEC60730_Class_C.mot**
File type	**S record file**
Offset	HEX 0
CPU Reset after download	Yes

Figure 4-4 CS+ Download file for Debug Tool Configuration

4.1.5 Code Generation (Design Tool)



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

EVDD setting: 4.0 V ≤ EVDD0 ≤ 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)

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

Frequency: - (kHz)

Divider: x1

fIHP: 32 (MHz)

fMAIN: 32 (MHz)

fCLK: 32000 (kHz)

fIMP: - (MHz)

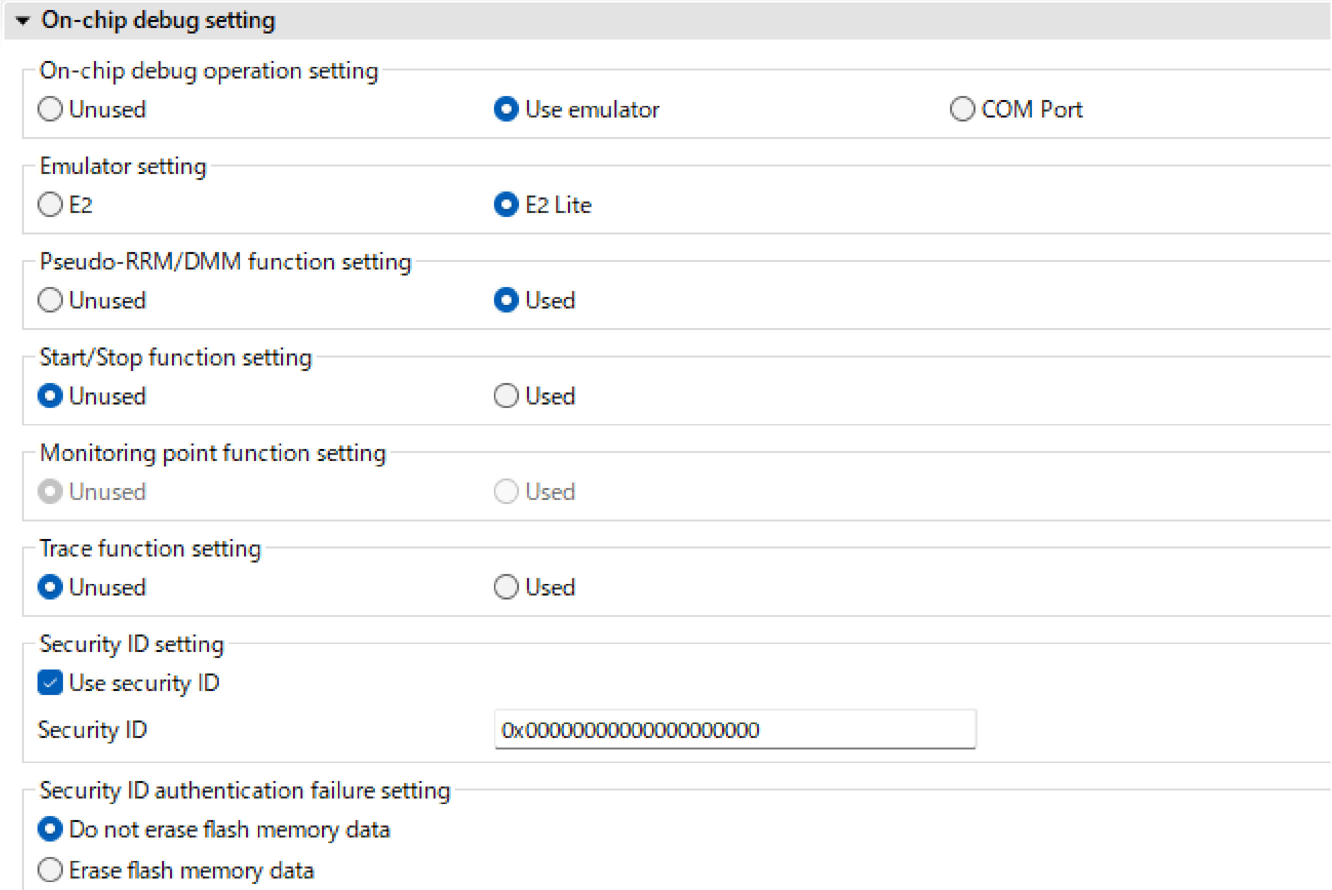
fMXP: - (MHz)

fIL: 32.768 (kHz)

fSXP: 32.768 (kHz)

fSXR: - (kHz)

Figure 4-5 Clock Generation Circuit



On-chip debug setting

On-chip debug operation setting

○ Unused ● Use emulator ○ COM Port

Emulator setting

○ E2 ● E2 Lite

Pseudo-RRM/DMM function setting

○ Unused ● Used

Start/Stop function setting

● Unused ○ Used

Monitoring point function setting

● Unused ○ Used

Trace function setting

● Unused ○ Used

Security ID setting

☑ Use security ID

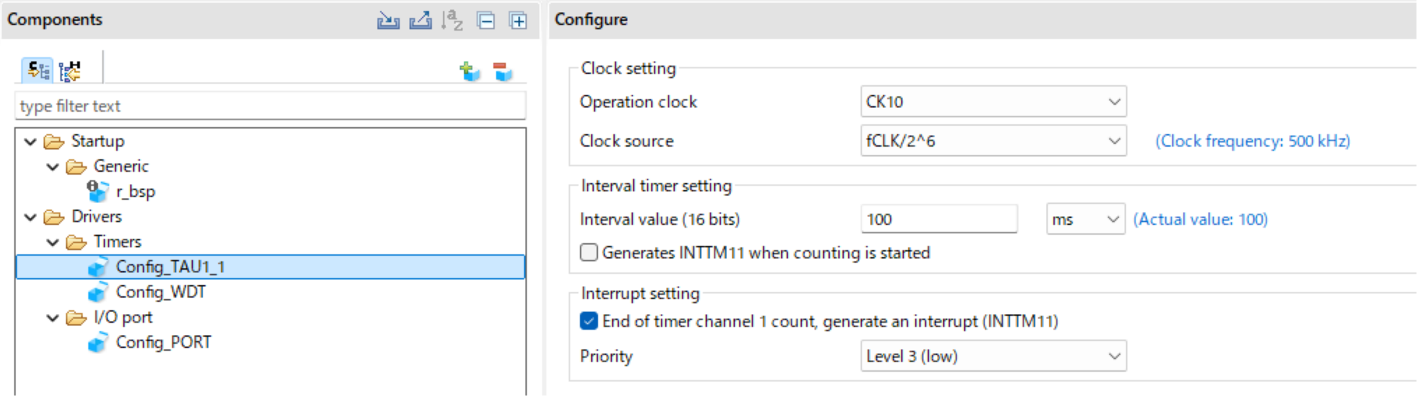
Security ID: 0x00000000000000000000

Security ID authentication failure setting

● Do not erase flash memory data

○ Erase flash memory data

Figure 4-6 System Setting



Components

Configure

Clock setting

Operation clock: CK10

Clock source: fCLK/2^6 (Clock frequency: 500 kHz)

Interval timer setting

Interval value (16 bits): 100 ms (Actual value: 100)

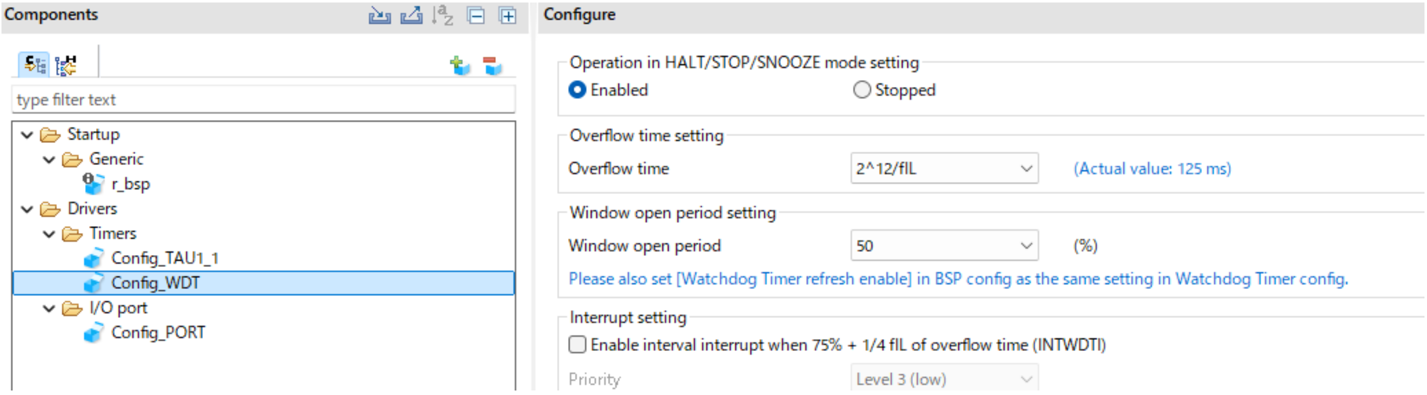
☐ Generates INTTM11 when counting is started

Interrupt setting

☑ End of timer channel 1 count, generate an interrupt (INTTM11)

Priority: Level 3 (low)

Figure 4-7 Periodic Timer Setting



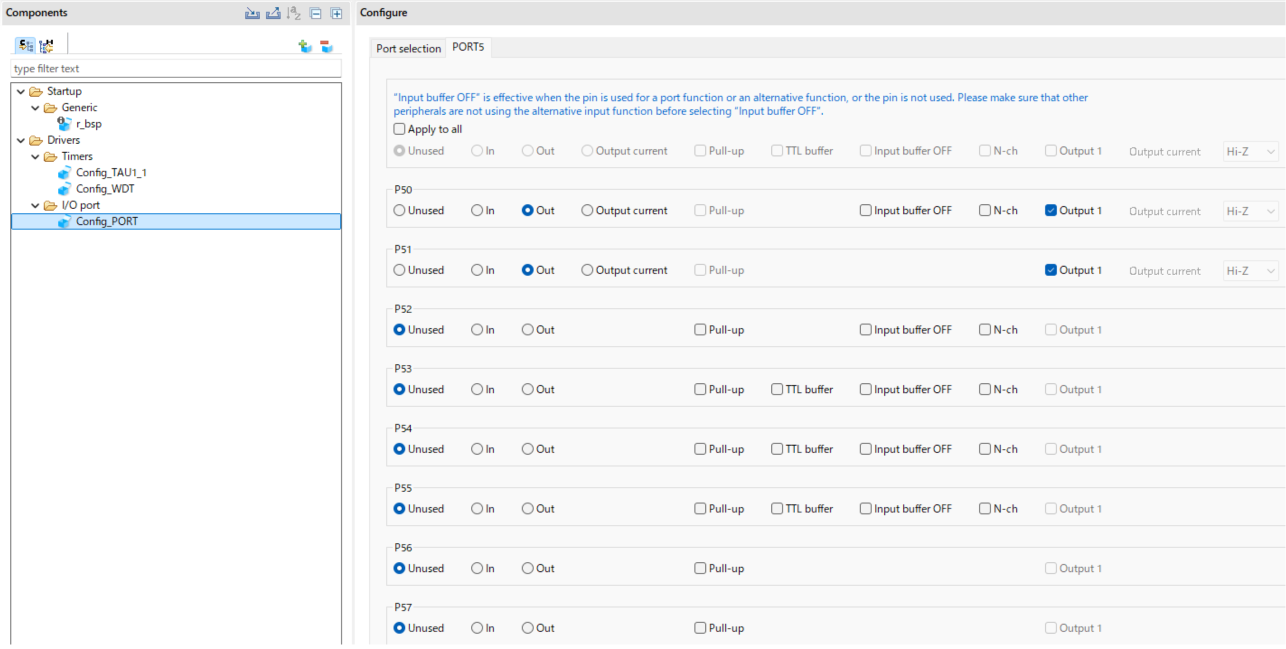
Operation in HALT/STOP/SNOOZE mode setting
● Enabled ○ Stopped

Overflow time setting
Overflow time: 2^12/fIL (Actual value: 125 ms)

Window open period setting
Window open period: 50 (%)
Please also set [Watchdog Timer refresh enable] in BSP config as the same setting in Watchdog Timer config.

Interrupt setting
☐ Enable interval interrupt when 75% + 1/4 fIL of overflow time (INTWDTI)
Priority: Level 3 (low)

Figure 4-8 WDT Setting



"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 ○ Output current ☐ Pull-up ☐ TTL buffer ☐ Input buffer OFF ☐ N-ch ☐ Output 1 Output current Hi-Z

P50: ○ Unused ○ In ● Out ○ Output current ☐ Pull-up ☐ Input buffer OFF ☐ N-ch ☑ Output 1 Output current Hi-Z

P51: ○ Unused ○ In ● Out ○ Output current ☐ Pull-up ☑ Output 1 Output current Hi-Z

P52: ● Unused ○ In ○ Out ☐ Pull-up ☐ Input buffer OFF ☐ N-ch ☐ Output 1

P53: ● Unused ○ In ○ Out ☐ Pull-up ☐ TTL buffer ☐ Input buffer OFF ☐ N-ch ☐ Output 1

P54: ● Unused ○ In ○ Out ☐ Pull-up ☐ TTL buffer ☐ Input buffer OFF ☐ N-ch ☐ Output 1

P55: ● Unused ○ In ○ Out ☐ Pull-up ☐ TTL buffer ☐ Input buffer OFF ☐ N-ch ☐ Output 1

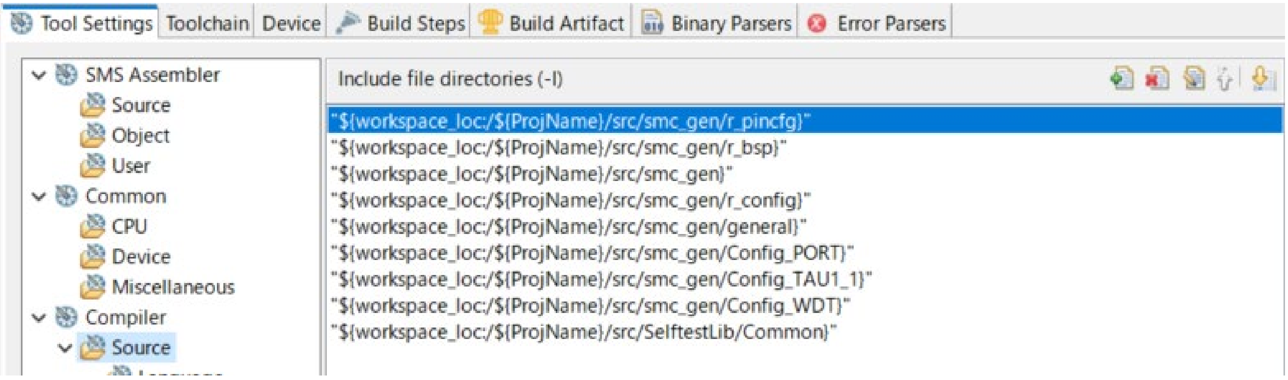
P56: ● Unused ○ In ○ Out ☐ Pull-up ☐ Output 1

P57: ● Unused ○ In ○ Out ☐ Pull-up ☐ Output 1

Figure 4-9 Port Setting

4.2 e^{2}stuido Settings

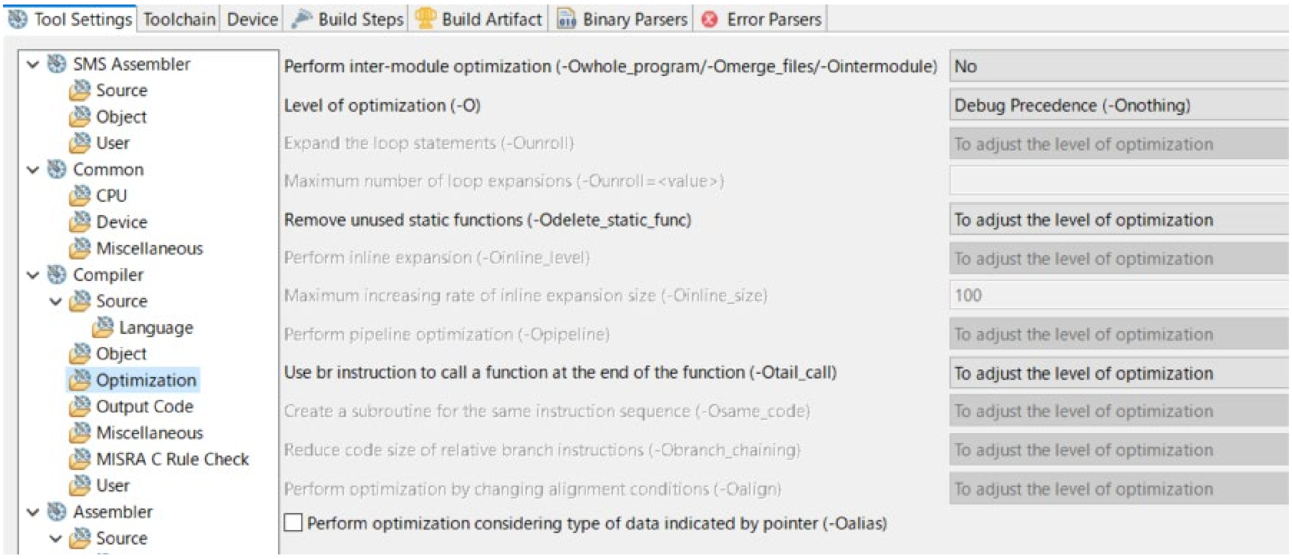
4.2.1 Complier Options



Include file directories (-I)

```
"${workspace_loc:/${ProjName}/src/smc_gen/r_pincfg}"
"${workspace_loc:/${ProjName}/src/smc_gen/r_bsp}"
"${workspace_loc:/${ProjName}/src/smc_gen}"
"${workspace_loc:/${ProjName}/src/smc_gen/r_config}"
"${workspace_loc:/${ProjName}/src/smc_gen/general}"
"${workspace_loc:/${ProjName}/src/smc_gen/Config_PORT}"
"${workspace_loc:/${ProjName}/src/smc_gen/Config_TAU1_1}"
"${workspace_loc:/${ProjName}/src/smc_gen/Config_WDT}"
"${workspace_loc:/${ProjName}/src/SelftestLib/Common}"
```

Figure 4-10 C Source Include Path

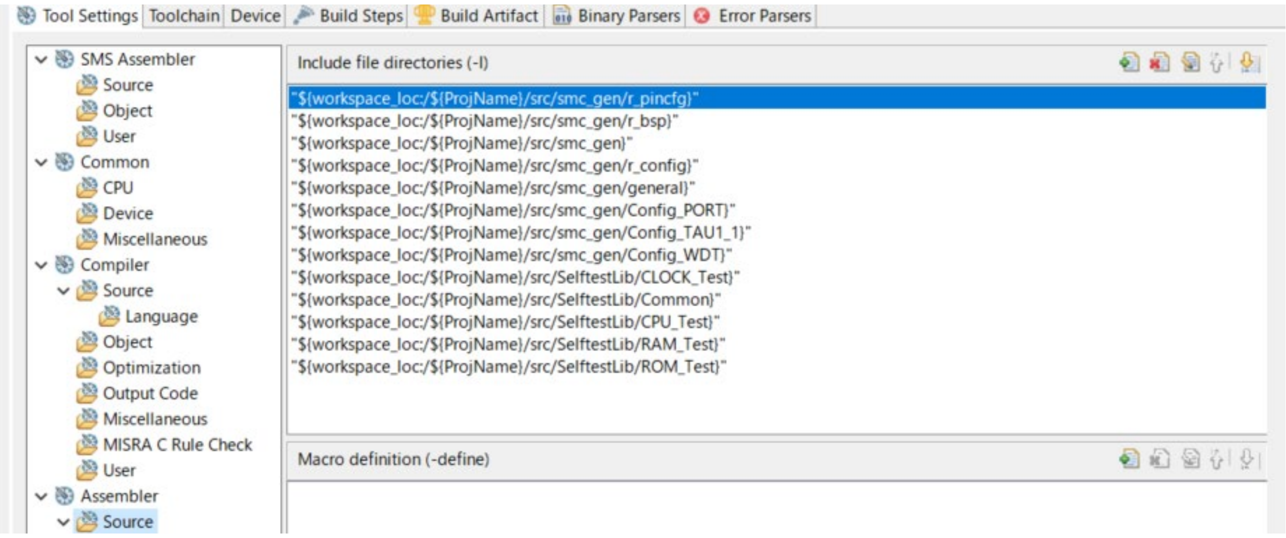


Perform inter-module optimization (-Owhole_program/-Omerge_files/-Ointermodule)	No
Level of optimization (-O)	Debug Precedence (-Onothing)
Expand the loop statements (-Ounroll)	To adjust the level of optimization
Maximum number of loop expansions (-Ounroll=<value>)	
Remove unused static functions (-Odelete_static_func)	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)	100
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



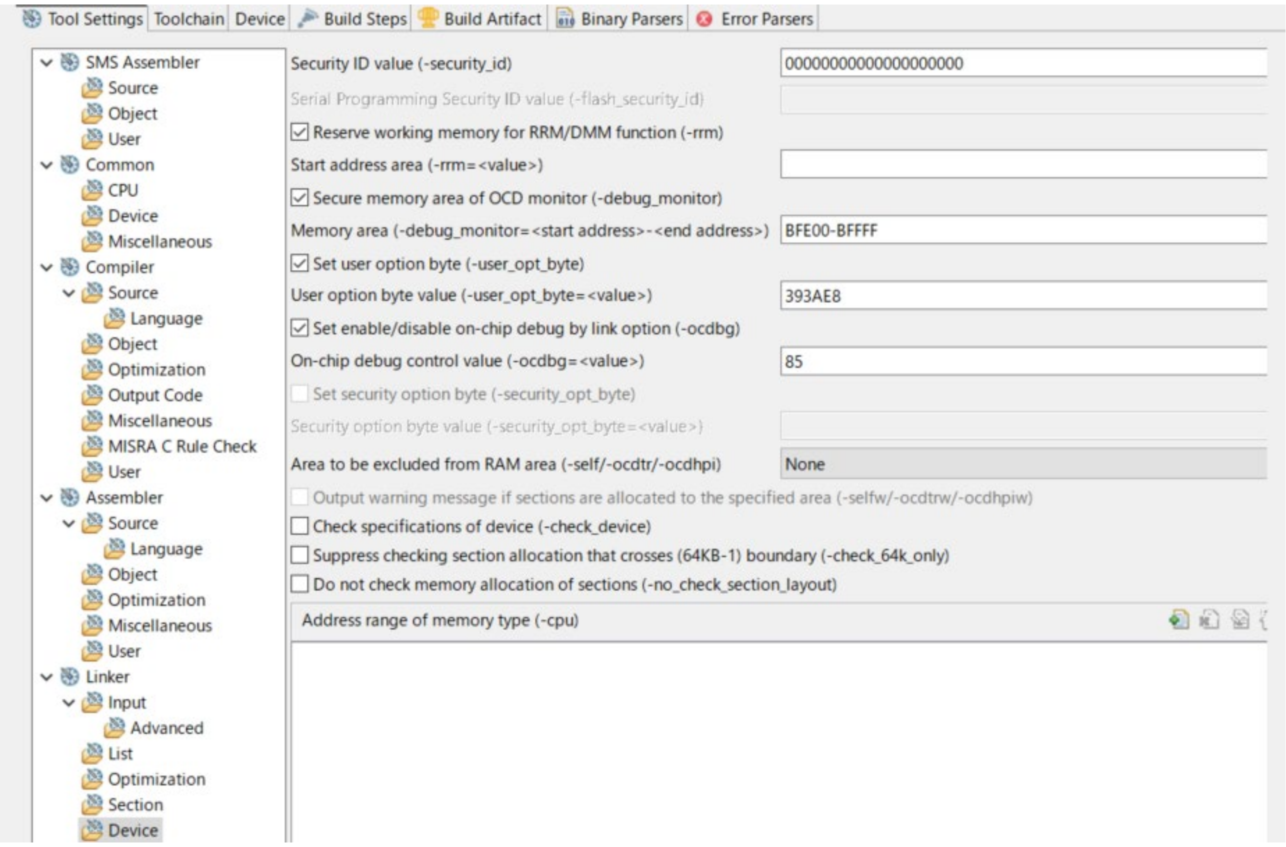
Include file directories (-I)

```
"${workspace_loc:/${ProjName}/src/smc_gen/r_pincfg}"
"${workspace_loc:/${ProjName}/src/smc_gen/r_bsp}"
"${workspace_loc:/${ProjName}/src/smc_gen}"
"${workspace_loc:/${ProjName}/src/smc_gen/r_config}"
"${workspace_loc:/${ProjName}/src/smc_gen/general}"
"${workspace_loc:/${ProjName}/src/smc_gen/Config_PORT}"
"${workspace_loc:/${ProjName}/src/smc_gen/Config_TAU1_1}"
"${workspace_loc:/${ProjName}/src/smc_gen/Config_WDT}"
"${workspace_loc:/${ProjName}/src/SelftestLib/CLOCK_Test}"
"${workspace_loc:/${ProjName}/src/SelftestLib/Common}"
"${workspace_loc:/${ProjName}/src/SelftestLib/CPU_Test}"
"${workspace_loc:/${ProjName}/src/SelftestLib/RAM_Test}"
"${workspace_loc:/${ProjName}/src/SelftestLib/ROM_Test}"
```

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>) BFE00-BFFFF

☑ Set user option byte (-user_opt_byte)

User option byte value (-user_opt_byte=<value>) 393AE8

☑ 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)

☐ Check specifications of device (-check_device)

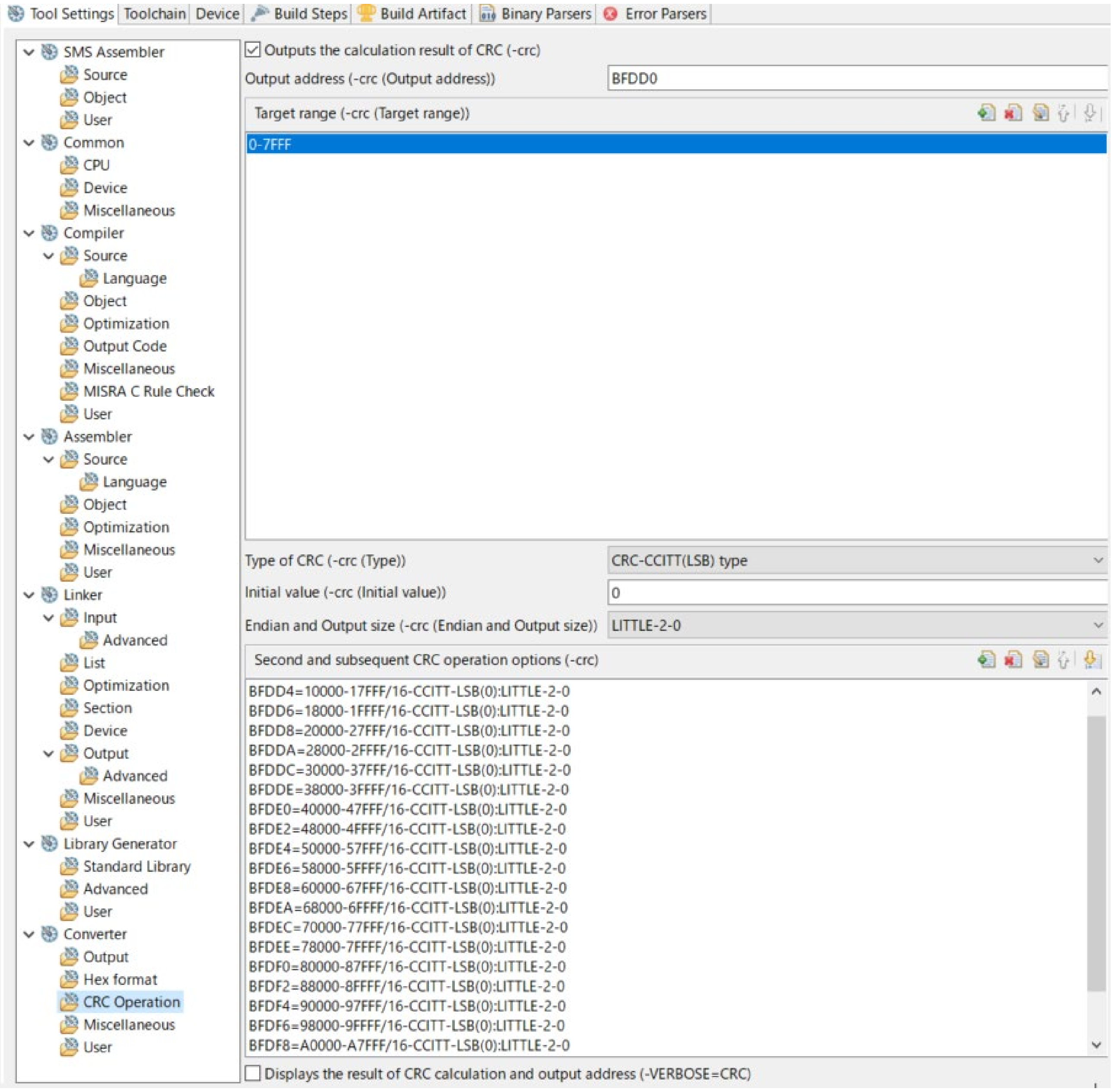
☐ Suppress checking section allocation that crosses (64KB-1) boundary (-check_64k_only)

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

Address range of memory type (-cpu)

Figure 4-13 Device Setting

4.2.4 Converter Options



☑ Outputs the calculation result of CRC (-crc)

Output address (-crc (Output address)): BFDD0

Target range (-crc (Target range)): 0-7FFF

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)

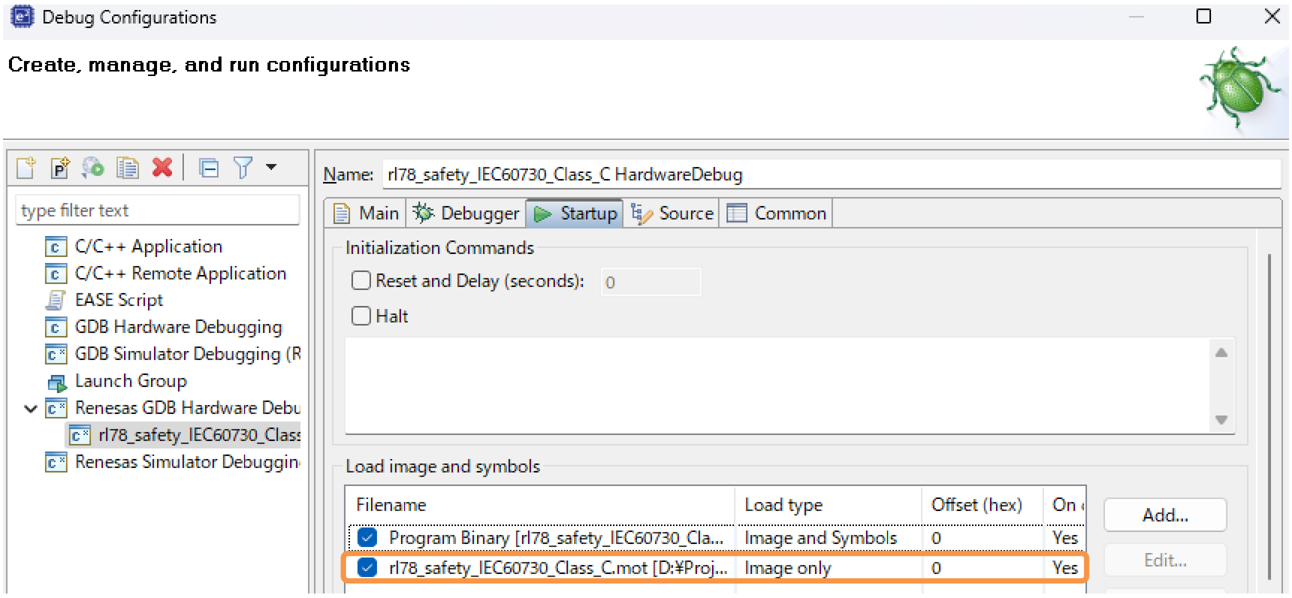
```
BFDD4=10000-17FFF/16-CCITT-LSB(0):LITTLE-2-0
BFDD6=18000-1FFFF/16-CCITT-LSB(0):LITTLE-2-0
BFDD8=20000-27FFF/16-CCITT-LSB(0):LITTLE-2-0
BFDDA=28000-2FFFF/16-CCITT-LSB(0):LITTLE-2-0
BFDDC=30000-37FFF/16-CCITT-LSB(0):LITTLE-2-0
BFDDE=38000-3FFFF/16-CCITT-LSB(0):LITTLE-2-0
BFDE0=40000-47FFF/16-CCITT-LSB(0):LITTLE-2-0
BFDE2=48000-4FFFF/16-CCITT-LSB(0):LITTLE-2-0
BFDE4=50000-57FFF/16-CCITT-LSB(0):LITTLE-2-0
BFDE6=58000-5FFFF/16-CCITT-LSB(0):LITTLE-2-0
BFDE8=60000-67FFF/16-CCITT-LSB(0):LITTLE-2-0
BFDEA=68000-6FFFF/16-CCITT-LSB(0):LITTLE-2-0
BFDEC=70000-77FFF/16-CCITT-LSB(0):LITTLE-2-0
BFDEE=78000-7FFFF/16-CCITT-LSB(0):LITTLE-2-0
BFDF0=80000-87FFF/16-CCITT-LSB(0):LITTLE-2-0
BFDF2=88000-8FFFF/16-CCITT-LSB(0):LITTLE-2-0
BFDF4=90000-97FFF/16-CCITT-LSB(0):LITTLE-2-0
BFDF6=98000-9FFFF/16-CCITT-LSB(0):LITTLE-2-0
BFDF8=A0000-A7FFF/16-CCITT-LSB(0):LITTLE-2-0
```

☐ Displays the result of CRC calculation and output address (-VERBOSE=CRC)

Figure 4-14 CRC Calculation Setting

Set the output destination address and calculation range according to the MCU

4.2.5 Debug Configurations



Name: rl78_safety_IEC60730_Class_C HardwareDebug

Initialization Commands

☐ Reset and Delay (seconds): 0

☐ Halt

Load image and symbols

Filename	Load type	Offset (hex)	On
☑ Program Binary [rl78_safety_IEC60730_Cla...	Image and Symbols	0	Yes
☑ rl78_safety_IEC60730_Class_C.mot [D:¥Proj...	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	ROM size
CPU instruction decode test stl_RL78_InitialInstructionTest	-	600μs	562bytes
CPU instruction decode test stl_RL78_InstructionTest	-	300μs	12939bytes
CPU general-purpose register test stl_RL78_registertest	-	200μs	1126bytes
CPU register test - PSW stl_RL78_registertest_psw	-	1.343μs	43bytes
CPU register test - SP stl_RL78_registertest_stack	-	1.125μs	50bytes
CPU register test - CS stl_RL78_registertest_cs	-	1.031μs	43bytes
CPU register test - ES stl_RL78_registertest_es	-	1.031μs	41bytes
CPU register test - PC stl_RL78_registertest_pc	-	0.857μs	17bytes
Hardware CRC stl_RL78_peripheral_crc	1024 bytes	700us	73bytes
System RAM test stl_RL78_RamTest	32 bytes* 2	1.4ms	1305bytes
Initial RAM test stl_RL78_InitialRamTest	508 bytes	10.8ms	980bytes
Hardware clock test stl_RL78_hw_clocktest	-	56.40μs	136bytes

6. 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（R01AN1062J）

Home page and Support Contact

Renesas Electronics Home Page

http://www.renesas.com/index.jsp

Contact for inquiries

http://www.renesas.com/contact/

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	2025.5.31		First edition issued
1.01	2025.10.20		Add e^2studio version in Development Environment

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/.