

## RL78 Family

R01AN4823EJ0100

Rev. 1.00

IEC60730/60335 Self Test Library of CARL78 for RL78 MCU Extended

Mar. 11, 2019

### Introduction

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

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

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

1. Class A: Control functions, which are not intended to be relied upon for the safety of the equipment.

Examples: Room thermostats, humidity controls, lighting controls, timers, and switches.

2. Class B: Control functions, which are intended to prevent unsafe operation of the controlled equipment.

Examples: Thermal cut-offs and door locks for laundry equipment.

3. Class C: Control functions, which are intended to prevent special hazards

Examples: Automatic burner controls and thermal cut-outs for closed.

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

This Application Note provides guidelines of how to use flexible sample software routines to assist with compliance with IEC60730/60335 class B safety standards.

These software routines provided are designed to be used after the system power on, or reset condition and also during the application program execution. The end user has the flexibility of what routines are included and how to integrate these routines into their overall application system design. This document and the accompanying test harness code provide examples of how to do this.

Note. This document is based on the European Norm EN60335-1:2002/A1:2004 Annex R, in which the Norm IEC 60730-1 (EN60730-1:2000) is used in some points. The Annex R of the mentioned Norm contains just a single sheet that jumps to the IEC 60730-1 for definitions, information and applicable paragraphs.

### Target Devices

RL78/G14 Microcontroller

## Contents

1. Self Test Libraries Introduction .....	3
2. Self Test Library Functions .....	4
2.1. CPU Register Tests.....	4
2.2. Invariable Memory Test – Flash ROM .....	12
2.3. Variable memory - SRAM .....	17
2.4. System Clock Test .....	27
3. Example Usage .....	32
3.1. CPU Verification .....	32
3.2. Flash ROM Verification .....	33
3.3. RAM Verification.....	34
3.4. System Clock Verification .....	35
3.5. Code Coverage .....	35
4. Benchmarking .....	36
4.1. Development Environment.....	36
4.2. CS+ Settings .....	36
4.3. Benchmark test results.....	39
5. Additional Hardware Resources.....	40
5.1. Additional Safety Functions.....	40
5.2. Additional Self Test Functions.....	45
6. Related Application Note.....	46
Revision Record .....	47
General Precautions in the Handling of MPU/MCU Products..エラー! ブックマークが定義されていません。	

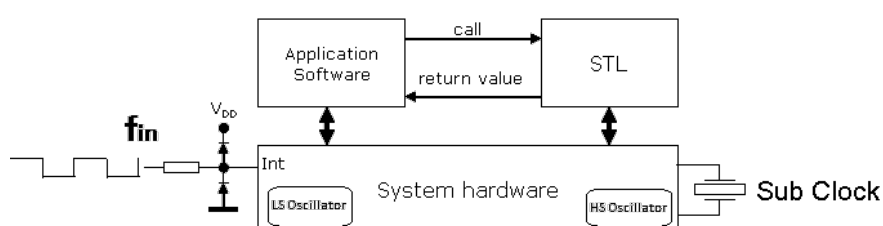
## 1. Self Test Libraries Introduction

The self test library (STL) provides self test functions covering the CPU registers, internal memory and system clock. The library test harness provides an Application Programmers Interface (API) for each of the self test modules, which are described in this applications note. These can be used in customer's application wherever required.

For the purposes of VDE certification, the self test library functions are built as separate modules. The **CS+** test harness allows each of the tests functions to be selected in turn and run as a stand alone function.

The system hardware requirements include that at least two independent clock sources are available, e.g. Crystal / ceramic oscillator and an independent oscillator or external input source. The requirement is needed to provide an independent clock reference for monitoring the system clock. The RL78 is able to provide these using the High speed and Low speed internal oscillators which are independent of each other.

Equally the application can provide a more accurate external reference clock or external crystal/resonators for the main system clock can equally be used.



**Figure 1 Self Test Library (STL) Configuration**

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

- CPU Registers

The following CPU registers tests are included in this library

All CPU working Registers in all four register banks, Stack Pointer (SP), Processor Status word (PSW), Extension registers ES and CS, Program Counter (PC).

Internal data path are verified as part of the correct operation of these register tests

IEC Reference - IEC 60730: 1999+A1:2003 Annex H - Table H.11.12.1 CPU.

- Invariable Memory

This tests the MCU internal Flash memory

IEC Reference - IEC 60730: 1999+A1:2003 Annex H – H2.19.4.1 CRC – Single Word.

- Variable Memory

This tests the Internal SRAM memory

IEC Reference - IEC 60730: 1999+A1:2003 Annex H – H2.19.4.1 CRC – Single Word.

- 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: 1999+A1:2003 Annex H – H2.19.4.1 CRC – Single Word.

## 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 following CPU registers are tested:

- **Working registers and Accumulator:** AX, HL, DE, BC in Register Banks 0 – 3

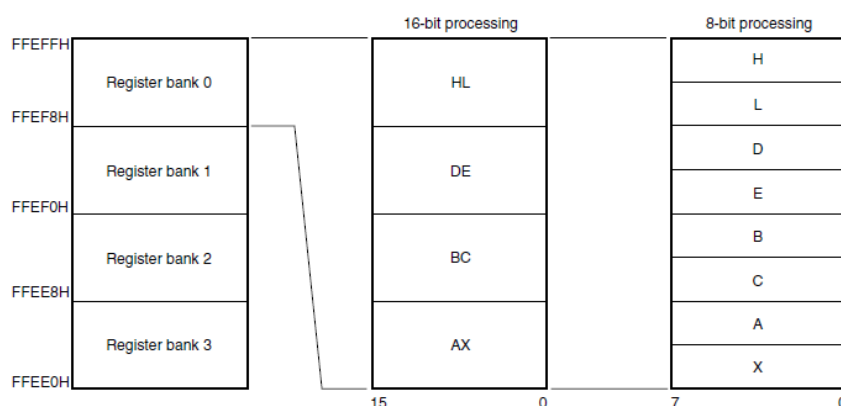


Figure 2 Working Register Configuration

- **Stack Pointer (SP)**

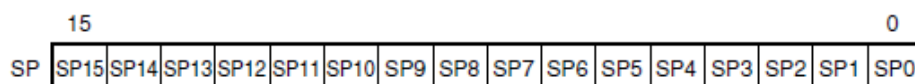


Figure 3 Stack Pointer Configuration

- **Processor Status Word (PSW)**

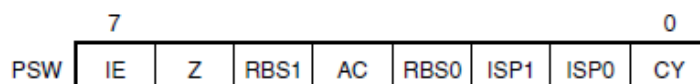


Figure 4 PSW Register Configuration

- Code Address Extension Register (CS)

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

Figure 5 Working 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 6 Working Register Configuration

- Program Counter (PC)

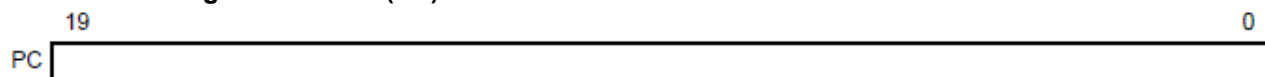


Figure 7 Program Counter Configuration

## 2.1.1. CPU Register Tests - Software API

Table 1: Source files: CPU Working Registers Tests

STL File name	Header Files
<a href="#">stl_RL78_registertest.asm</a>	None
Test Harness File Names	Header Files
<a href="#">main.c</a>	<a href="#">stl.h</a>
<a href="#">stl_global_data_example.c</a>	<a href="#">main.h</a>
<a href="#">stl_main_example_support function.c</a>	<a href="#">stl_gobal_data_example.h</a>
<a href="#">stl_peripheralinit.c</a>	

Syntax	
char stl_RL78_registertest(void)	
Description	
<p>This module tests the RL78 working registers and accumulators.</p> <p>Registers AX, HL, DE, BC in all three register banks (Banks 0, 1, 2, 3)</p> <p>These registers are tested as 16bit registers.</p> <p>The following tests are performed for each register:</p> <ol style="list-style-type: none"> <li>1. Write h'5555 to the register being tested.</li> <li>2. Read back and check they are equal.</li> <li>3. Write h'AAAA to the register being tested.</li> <li>4. Read back and check they are equal.</li> </ol> <p>It is the calling function's responsibility to ensure no interrupts occur during this test. In addition, Register Bank 0 (RB0) must be selected when this test starts.</p> <p>The original register contents are restored on completion of the test</p> <p>The function "<a href="#">indicate_test_result</a>" will be called by the test harness control files (<a href="#">main.c</a>) to process the test result</p> <p>Note: Function "<a href="#">indicate_test_result</a>" is located in the module <a href="#">stl_main_example_support function.c</a></p>	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
char	<p>Test result of CPU register C</p> <p>0 = Test passed. 1 = Test or parameter check failed</p>

Table 2: Source files: CPU Registers Tests – PSW

STL File name	Header Files
<a href="#">stl_RL78_registertest_psw.asm</a>	<a href="#">stl.h</a>
Test Harness File Names	Header Files
<a href="#">main.c</a> <a href="#">stl_global_data_example.c</a> <a href="#">stl_main_example_support function.c</a> <a href="#">stl_peripheralinit.c</a>	<a href="#">main.h</a> <a href="#">stl_gobal_data_example.h</a>

Syntax	
char stl_RL78_registertest_psw(void)	
Description	
<p>Test the 8bit Processor Status Word (PSW) register</p> <p>The following tests are performed:</p> <ol style="list-style-type: none"> <li>1. Write h'55 to the register being tested.</li> <li>2. Read back and check it is equal.</li> <li>3. Write h'AA to the register being tested.</li> <li>4. Read back and check that it is equal.</li> </ol> <p>It is the calling function's responsibility to ensure no interrupts occur during this test.</p> <p>The original register content is restored on completion of the test</p> <p>The function "<a href="#">indicate_test_result</a>" will be called by the test harness control files (<a href="#">main.c</a>) to process the test result</p> <p>Note: Function "<a href="#">indicate_test_result</a>" is located in the module <a href="#">stl_main_example_support function.c</a></p>	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
char	<p>Test result of CPU register C</p> <p>0 = Test passed. 1 = Test or parameter check failed</p>

Table 3: Source files: CPU Registers Tests - SP

STL File name	Header Files
<a href="#">stl_RL78_registertest_stack.asm</a>	<a href="#">stl.h</a>
Test Harness File Names	Header Files
<a href="#">main.c</a> <a href="#">stl_global_data_example.c</a> <a href="#">stl_main_example_support function.c</a> <a href="#">stl_peripheralinit.c</a>	<a href="#">main.h</a> <a href="#">stl_gobal_data_example.h</a>

Syntax	
char stl_RL78_registertest_stack(void)	
Description	
<p>Test the 16bit Stack Pointer (SP) register</p> <p>The following tests are performed:</p> <ol style="list-style-type: none"> <li>1. Write h'5555 to the register being tested.</li> <li>2. Read back and check it is equal.</li> <li>3. Write h'AAAA to the register being tested.</li> <li>4. Read back and check that it is equal.</li> </ol> <p>It is the calling function's responsibility to ensure no interrupts occur during this test.</p> <p>The original register content is restored on completion of the test</p> <p>The function "<a href="#">indicate_test_result</a>" will be called by the test harness control files (<a href="#">main.c</a>) to process the test result</p> <p>Note: Function "<a href="#">indicate_test_result</a>" is located in the module <a href="#">stl_main_example_support function.c</a></p>	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
char	<p>Test result of CPU register C</p> <p>0 = Test passed. 1 = Test or parameter check failed</p>



Table 4: Source files: CPU Registers Tests - CS

STL File name	Header Files
<a href="#">stl_RL78_registertest_cs.asm</a>	<a href="#">stl.h</a>
Test Harness File Names	Header Files
<a href="#">main.c</a> <a href="#">stl_global_data_example.c</a> <a href="#">stl_main_example_support function.c</a> <a href="#">stl_peripheralinit.c</a>	<a href="#">main.h</a> <a href="#">stl_gobal_data_example.h</a>

Syntax	
char stl_RL78_registertest_cs(void)	
Description	
<p>Test the 8bit code extension (CS) register</p> <p>The following tests are performed:</p> <ol style="list-style-type: none"> <li>1. Write h'05 to the register being tested.</li> <li>2. Read back and check it is equal.</li> <li>3. Write h'0A to the register being tested.</li> <li>4. Read back and check that it is equal.</li> </ol> <p>Please note that the top 4 bit are fixed to “0”</p> <p>It is the calling function’s responsibility to ensure no interrupts occur during this test.</p> <p>The original register content is restored on completion of the test</p> <p>The function “<a href="#">indicate_test_result</a>” will be called by the test harness control files (<a href="#">main.c</a>) to process the test result</p> <p>Note: Function “<a href="#">indicate_test_result</a>” is located in the module <a href="#">stl_main_example_support function.c</a></p>	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
char	<p>Test result of CPU register C</p> <p>0 = Test passed. 1 = Test or parameter check failed</p>

Table 5: Source files: CPU Registers Tests - ES

STL File name	Header Files
<a href="#">stl_RL78_registertest_es.asm</a>	<a href="#">stl.h</a>
Test Harness File Names	Header Files
<a href="#">main.c</a> <a href="#">stl_global_data_example.c</a> <a href="#">stl_main_example_support function.c</a> <a href="#">stl_peripheralinit.c</a>	<a href="#">main.h</a> <a href="#">stl_gobal_data_example.h</a>

Syntax	
char stl_RL78_registertest_es(void)	
Description	
<p>Test the 8bit data extension (ES) register</p> <p>The following tests are performed:</p> <ol style="list-style-type: none"> <li>1. Write h'05 to the register being tested.</li> <li>2. Read back and check it is equal.</li> <li>3. Write h'0A to the register being tested.</li> <li>4. Read back and check that it is equal.</li> </ol> <p>Please note that the top 4 bit are fixed to “0”</p> <p>It is the calling function’s responsibility to ensure no interrupts occur during this test.</p> <p>The original register content is restored on completion of the test</p> <p>The function “<a href="#">indicate_test_result</a>” will be called by the test harness control files (<a href="#">main.c</a>) to process the test result</p> <p>Note: Function “<a href="#">indicate_test_result</a>” is located in the module <a href="#">stl_main_example_support function.c</a></p>	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
char	<p>Test result of CPU register C</p> <p>0 = Test passed. 1 = Test or parameter check failed</p>

Table 6: Source files: CPU Registers Tests - PC

STL File name	Header Files
<a href="#">stl_RL78_registertest_pc.asm</a>	<a href="#">stl.h</a>
Test Harness File Names	Header Files
<a href="#">main.c</a>	<a href="#">main.h</a>

Syntax	
char stl_RL78_registertest_pc(void)	
Description	
<p>Test the program counter (PC) register</p> <p>The following tests are performed:</p> <ol style="list-style-type: none"> <li>1. Call the program counter (PC) test function with call instruction.</li> <li>2. The test function sets return address saved on the stack in the L register · DE register and returns.</li> <li>3. After calling the test function with the call instruction, confirm that the address (PC) of the instruction placed next to the call instruction is equal to the return value (L - DE).</li> </ol> <p>The first 4 bits of the L register are fixed "0" values.</p> <p>It is the calling function's responsibility to ensure no interrupts occur during this test.</p> <p>The original register content is restored on completion of the test.</p> <p>The function "<a href="#">indicate_test_result</a>" will be called by the test harness control files (<a href="#">main.c</a>) to process the test result</p> <p>Note: Function "<a href="#">indicate_test_result</a>" is located in the module <a href="#">stl_main_example_support function.c</a></p>	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
char	<p>Test result of CPU register C</p> <p>0 = Test passed. 1 = Test or parameter check failed</p>

## 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 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. Both the hardware and software self test functions are able to be executed iteratively, 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), a starting 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 for calculating the reference CRC value. Here a compatible CRC calculation routine is provided as part of the test harness for reference.

### 2.2.1. CRC16-CCITT Algorithm

The RL78 includes a CRC module that includes support for the CRC16-CCITT. Using this software to drive the CRC module produces this 16-bit CRC16-CCITT:

#### Software Algorithm

- CCITT 16 Polynomial =  $0x1021 (x^{16} + x^{12} + x^5 + 1)$
- Input Data Width = 8 bits
- Data Input = Not Bit Reversed
- Initial value = 0x0000 or 16 bit previous partial result
- Calculated Result = 16 bits (not bit reversed)

#### Hardware Algorithm

- CCITT 16 Polynomial =  $0x1021 (x^{16} + x^{12} + x^5 + 1)$
- Input Data Width = 8 bits
- Data Input = Bit Reversed
- Initial value = 0x0000 or 16 bit previous partial result
- Calculated Result = 16 bits (Bit reversed)

**2.2.2. Software CRC - Software API**

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

**Table 7: Source files: Software CRC**

STL File name	Header Files
<a href="#">stl_RL78_sw_crc.asm</a>	<a href="#">stl.h</a>
Test Harness File Names	Header Files
<a href="#">main.c</a>	<a href="#">main.h</a>
<a href="#">stl_global_data_example.c</a>	<a href="#">stl_gobal_data_example.h</a>
<a href="#">stl_main_example_support function.c</a>	
<a href="#">stl_peripheralinit.c</a>	

Syntax	
unsigned short stl_RL78_sw_crc_asm (unsigned short crc, CHECKSUM_CRC_TEST_AREA *p);	
Description	
<p>This function calculates a CRC value over the address range supplied using the software CRC calculation module. The start address and calculation range (Length) are passed by the calling function via the structure shown in the table below. The partial or full calculated result is returned for verification (if required) against the reference CRC value.</p> <p>The function “<a href="#">indicate_test_result</a>” will be called by the test harness control files (<a href="#">main.c</a>) to process the test result</p> <p>Note: Function “<a href="#">indicate_test_result</a>” is located in the module <a href="#">stl_main_example_support function.c</a></p>	
Input Parameters	
unsigned short crc	Value for starting the CRC calculation
CHECKSUM_CRC_TEST_AREA *p	Pointer to the structure where the start address and calculation range is located
Output Parameters	
NONE	N/A
Return Values	
Unsigned short	16 bit calculated CRC value (Full or partial result) CPU Register BC

**Source files: Software CRC Parameter Structure**

The following structure is implemented in the files [stl.h](#) and [main.c](#) and is used to provide calculation parameters for the CRC function.

Syntax	
static CHECKSUM_CRC_TEST_AREA checksum_crc;	
Description	
Structure declaration and instance providing the parameters to be passed to software CRC module ( <a href="#">stl_RL78_sw_crc.asm</a> ) by the calling function in <a href="#">main.c</a>	
Input Parameters	
Unsigned long length;	Range (length = number of bytes) of memory to be tested.
Unsigned long start_address	Start address for CRC calculation
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

## 2.2.3. Hardware CRC - Software API

Table 8: Source files: Hardware CRC Calculation

STL File name	Header Files
<a href="#">stl_RL78_peripheral_crc.asm</a>	<a href="#">&lt;ior5f100le.h&gt;</a> <a href="#">&lt;ior5f100le_ext.h&gt;</a> <a href="#">stl.h</a>
Test Harness File Names	Header Files
<a href="#">main.c</a> <a href="#">stl_global_data_example.c</a> <a href="#">stl_main_example_support function.c</a> <a href="#">stl_peripheralinit.c</a>	<a href="#">main.h</a> <a href="#">stl_gobal_data_example.h</a>

Syntax	
unsigned short stl_RL78_peripheral_crc(unsigned short gcr, CHECKSUM_CRC_TEST_AREA *p)	
Description	
<p>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.</p> <p>The function “<a href="#">indicate_test_result</a>” will be called by the test harness control files (<a href="#">main.c</a>) to process the test result</p> <p>Note:Function “<a href="#">indicate_test_result</a>” is located in the module <a href="#">stl_main_example_support function.c</a></p>	
Input Parameters	
unsigned short gcr	Value for starting the CRC calculation
CHECKSUM_CRC_TEST_AREA *p	Pointer to the structure where the start address and calculation range is located
Output Parameters	
NONE	N/A
Return Values	
unsigned short	16 bit calculated CRC value (Full or partial result) CPU Register BC

## Source files: Hardware CRC Parameter Structure

Syntax	
static CHECKSUM_CRC_TEST_AREA checksum_crc;	
Description	
Structure declaration and instance providing the parameters to be passed to the hardware CRC module ( <a href="#">stl_RL78_peripheral_crc.asm</a> ) by the calling function in <a href="#">main.c</a> . Note: This is the same structure as used by the software CRC function.	
Input Parameters	
unsigned int length;	Range (length = number of bytes) of memory to be tested.
unsigned int start_address	Start address for CRC calculation
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A



## 2.3. Variable memory - SRAM

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

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

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

The algorithms themselves are destructive (they do not preserve the current RAM values). It is the user's responsibility to preserve the Ram contents during testing after the application system has been initialised or while in operation. The system March C and March X test modules are designed such that small parts of the Ram area can be tested, thus minimising the need to provide a large temporary area to save the data under test. Additional version of the test module ("stl\_RL78\_march\_c\_initial" and "stl\_RL78\_march\_x\_initial"), are included that are designed to run before the system has been initialised, so that the complete memory area can be tested before starting the main application.

As the area of RAM being tested can not be used for anything else while it is being tested, making the testing of RAM used for the stack particularly difficult. Practically this area can only be tested before the application C-Stack is initialised or after the application operation is complete.

The following section introduces the specific March Tests.

### 2.3.1. Algorithms

#### 1) March C

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

1. Stuck At Faults (SAF)
  - The logic value of a cell or a line is always 0 or 1.
2. Transition Faults (TF)
  - A cell or a line that fails to undergo a 0→1 or a 1→0 transition.
3. Coupling Faults (CF)
  - A write operation to one cell changes the content of a second cell.
4. Address Decoder Faults (AF)
  - Any fault that affects 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. Write all zeros to array ( <(w0) )
2. Starting at lowest address, read zeros, write ones, increment up array bit by bit. ( >(r0,w1) )
3. Starting at lowest address, read ones, write zeros increment up array bit by bit. ( >(r0,w1) )
4. Starting at highest address, read zeros, write ones, decrement down array bit by bit. ( <(r0,w1) )
5. Starting at highest address, read ones, write zeros, decrement down array bit by bit. ( <(r1,w0) )
6. Read all zeros from array. ( <(r0) )

### 2) March X

The March X algorithm is a simpler and therefore faster algorithm, but not as thorough as it consists of only four March elements with a total of four operations

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

These are the 4 March elements:-

1. Write all zeros to array (  $\diamond(w0)$  )
2. Starting at lowest address, read zeros, write ones, increment up array bit by bit. (  $\triangleright(r0,w1)$  )
3. Starting at highest address, read ones, write zeros, decrement down array bit by bit. (  $\triangleleft(r1,w0)$  )
4. Read all zeros from array. (  $\diamond(r0)$  )

### 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 a single byte to be tested. However, for all faults types to be detected it is important to test a data range bigger than one byte.

**Table 9: Source files: System March C**

STL File name	Header Files
<a href="#">stl_RL78_march_c.asm</a>	<a href="#">stl.h</a>
Test Harness File Names	Header Files
<a href="#">main.c</a>	<a href="#">main.h</a>
<a href="#">stl_global_data_example.c</a>	<a href="#">stl_gobal_data_example.h</a>
<a href="#">stl_main_example_support function.c</a>	
<a href="#">stl_peripheralinit.c</a>	

Declaration	
char stl_RL78_march_c(unsigned char __far *addr, unsigned short num)	
Description	
<p>This function tests the Ram memory using the March C algorithm over the address range supplied by the calling function. The result status (Pass / Fail) is returned. This module is designed to be executed after the application system has been initialised.</p> <p>The function “<a href="#">indicate_test_result</a>” will be called by the test harness control files (<a href="#">main.c</a>) to process the test result</p> <p>Note: Function “<a href="#">indicate_test_result</a>” is located in the module <a href="#">stl_main_example_support function.c</a></p>	
Input Parameters	
unsigned char __far *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	<p>Test status result contained in CPU register C</p> <p>0 = Test passed. 1 = Test or parameter check failed</p>

**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 a single byte to be tested. However, for all faults types to be detected it is important to test a data range bigger than one byte.

**Table 10: Source files:**

STL File name	Header Files
<a href="#">stl_RL78_march_x.asm</a>	<a href="#">stl.h</a>
Test Harness File Names	Header Files
<a href="#">main.c</a>	<a href="#">main.h</a>
<a href="#">stl_global_data_example.c</a>	<a href="#">stl_gobal_data_example.h</a>
<a href="#">stl_main_example_support function.c</a>	
<a href="#">stl_peripheralinit.c</a>	

Declaration	
char stl_RL78_march_x(unsigned char __far *addr, unsigned short num)	
Description	
<p>This function tests the Ram memory using the March X algorithm over the address range supplied by the calling function. The result status (Pass / Fail) is returned. This module is designed to be executed after the application system has been initialised.</p> <p>The function “<a href="#">indicate_test_result</a>” will be called by the test harness control files (<a href="#">main.c</a>) to process the test result</p> <p>Note: Function “<a href="#">indicate_test_result</a>” is located in the module <a href="#">stl_main_example_support function.c</a></p>	
Input Parameters	
unsigned char __far *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	<p>Test result of CPU register C</p> <p>0 = Test passed. 1 = Test or parameter check failed</p>

### 2.3.2.3. Initial March C

The initial March C test is designed to run before the application system has been initialised and is executed without using function calls from the test harness. Entry to the self test is made by a “jump” from the modified “[startup.asm](#)” module and return to “[startup.asm](#)” module is also made with a “jump”. The test status result is contained in the 8bit accumulator (A). Therefore this module is designed to provide a complete RAM test before the system is started and the “C” environment is initialised.

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

**Table 11: Source files: Initial March C**

STL File name	Header Files
<a href="#">stl_RL78_march_c_initial.asm</a>	None
Test Harness File Names	Header Files
<a href="#">startup.asm</a>	None

Declaration	
stl_RL78_march_c_initial	
Description	
<p>This function tests the Ram memory using the March C algorithm over the address range supplied by the calling function. The result status (Pass / Fail) is returned. This module is designed to be executed before the application system has been initialised and therefore does not use any function calls.</p> <p>The function “<a href="#">indicate_test_result</a>” will be called by the test harness control files (<a href="#">main.c</a>) to process the test result  Note: Function “<a href="#">indicate_test_result</a>” is located in the module <a href="#">stl_main_example_support function.c</a></p>	
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	<p>Test status result</p> <p>0 = Test passed. 1 = Test or parameter check failed</p>

### 2.3.2.4. Initial March X

The initial March C test is designed to run before the application system has been initialised and is executed without using function calls from the test harness. Entry to the self test is made by a “jump” from the modified “[startup.asm](#)” module and return to “[startup.asm](#)” module is also made with a “jump”. The test status result is contained in the 8bit accumulator (A). Therefore this module is designed to provide a complete RAM test before the system is started and the “C” environment is initialised.

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

**Table 12: Source files: Initial March X**

STL File name	Header Files
<a href="#">stl_RL78_march_x_initial.asm</a>	None
Test Harness File Names	Header Files
<a href="#">startup.asm</a>	None

Declaration	
stl_RL78_march_x_initial	
Description	
<p>This function tests the Ram memory using the March X algorithm over the address range supplied by the calling function. The result status (Pass / Fail) is returned. This module is designed to be executed before the application system has been initialised and therefore does not use any function calls.</p> <p>The function “<a href="#">indicate_test_result</a>” will be called by the test harness control files (<a href="#">main.c</a>) to process the test result  Note: Function “<a href="#">indicate_test_result</a>” is located in the module <a href="#">stl_main_example_support function.c</a></p>	
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	<p>Test result of CPU register A</p> <p>0 = Test passed. 1 = Test or parameter check failed.</p>

**2.3.2.5. Stack area test (March C)**

Use C stack resource to execute with normal function call from test harness. It is possible to test all of the STACK area. Since the test is destructive, test after saving the current state to the buffer. It is possible to partially test by switching the offset of the STACK\_TEST\_AREA parameter for each test.

RAM test is performed using System March C.

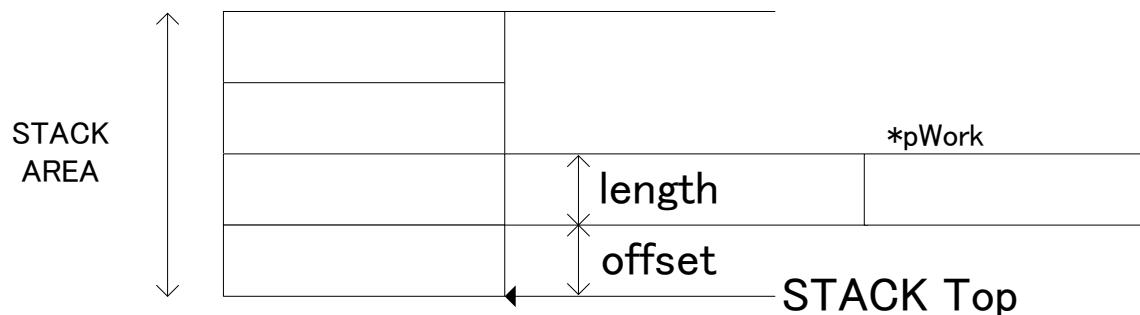
**Table 13: Source files: Stack area test (March C)**

STL File name	Header Files
<a href="#">stl_RL78_RamTest_Stacks_c.asm</a>	None
Test Harness File Names	Header Files
<a href="#">main.c</a>	<a href="#">main.h</a>

Declaration	
char stl_RL78_RamTest_Stacks_c(STACK_TEST_AREA *p)	
Description	
Switch the stack pointer (SP) to the specified area, test the address range of the specified buffer RAM using the March C algorithm, and if the result (pass / fail) is normal, the contents of the stack area to the buffer RAM. Next, we use the March C algorithm to test the stack area and restore the contents saved in the buffer RAM and the stack pointer (SP). And it returns the test result (pass / fail). This module is executed after initialization of the application system.	
The test harness control file ( <a href="#">main.c</a> ) calls the function " <a href="#">indicate_test_result</a> " to process the test result.	
Note: The function " <a href="#">indicate_test_result</a> " is in the module <a href="#">stl_main_example_support function.c</a> .	
Input Parameters	
STACK_TEST_AREA *p	Pointer to structure storing buffer RAM / size / new stack area
Output Parameters	
NONE	N/A
Return Values	
char	Test result of CPU register C 0 = Test passed. 1 = Test or parameter check failed.

Source files: Stack area test parameter structure

Declaration	
static STACK_TEST_AREA stack_test	
Description	
Structure declarations and instances that provide the parameters passed from the <a href="#">main.c</a> caller function to the stack area test module ( <a href="#">stl_RL78_RamTest_Stacks_c.asm</a> ).	
Note: This is the same as the structure of the <a href="#">stl_RL78_RamTest_Stacks_x</a> function.	
Input Parameters	
char *pWork;	Start address of the area to save the contents of the stack
unsigned short length	Size of test target
unsigned short offset	Stack area to be tested (offset from stack TOP)
char *pNewSp	Stack pointer to temporarily use during testing
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A





**2.3.2.6. Stack area test (March X)**

Use C stack resource to execute with normal function call from test harness. It is possible to test all of the STACK area. Since the test is destructive, test the current state after saving it to the buffer. You can partially test by switching the offset of the STACK\_TEST\_AREA parameter for each test.

RAM test is performed using System March C.

**Table 14: Source files: Stack area test (March X)**

STL File name	Header Files
<a href="#">stl_RL78_RamTest_Stacks_x.asm</a>	None
Test Harness File Names	Header Files
<a href="#">main.c</a>	<a href="#">main.h</a>

Syntax	
char stl_RL78_RamTest_Stacks_x(STACK_TEST_AREA *p)	
Description	
Switch the stack pointer (SP) to the specified area, test the address range of the specified buffer RAM using the March X algorithm, and if the result (pass / fail) is normal, the contents of the stack area To the buffer RAM. Next, we use the March X algorithm to test the stack area and restore the contents saved in the buffer RAM and the stack pointer (SP). And it returns the test result (pass / fail). This module is executed after initialization of the application system. The test harness control file ( <a href="#">main.c</a> ) calls the function " <a href="#">indicate_test_result</a> " to process the test result. Note: The function " <a href="#">indicate_test_result</a> " is in the module <a href="#">stl_main_example_support function.c</a> .	
Input Parameters	
STACK_TEST_AREA *p	Pointer to structure storing buffer RAM / size / new stack area
Output Parameters	
NONE	N/A
Return Values	
char	Test result of CPU register C 0 = Test passed. 1 = Test or parameter check failed

Source files: Stack area test parameter structure

Declaration	
static STACK_TEST_AREA stack_test	
Description	
Structure declarations and instances that provide the parameters passed from the main.c caller function to the stack area test module ( <a href="#">stl_RL78_RamTest_Stacks_x.asm</a> ).	
Note: This is the same as the structure of the <a href="#">stl_RL78_RamTest_Stacks_c</a> function.	
Input Parameters	
char *pWork;	Start address of the area to save the contents of the stack
unsigned short length	Size of test target
unsigned short offset	Stack area to be tested (offset from stack TOP)
char *pNewSp	Stack pointer to temporarily use during testing
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

## 2.4. System Clock Test

Two self test modules (hardware and software base) are provided for the RL78 self test library in order to be able to test the internal system clock (CPU and Peripheral clocks). The software measurement module is included for backward compatibility with previous products and also to allow for any RL78 devices where the Timer Array does not include the additional hardware capability, or that the timer is used by the application and is not available to be used as part of the MCU self tests. 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 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 if the operation frequency of the main clock deviates during runtime from a predefined range, then this can be detected by the system. The accuracy of the measurement obviously depends on the accuracy of the reference clock source. For example an external signal input or 32 KHz crystal can provide a more accurate measurement of the system clock than the internal low speed oscillator. This however does require the 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, in order to identify the appropriate fault state. Both the software and hardware measurement function use the same return status format.

The modules compare the measured (captured) time is within a reference window (upper and lower limit values) using the user defined reference values set in the “`stl_clocktest_h`” header file. This header file defines the reference values for both software and hardware measurements and also the input test port pin for the software measurement.

### 2.4.1. 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:-

- The internal Low-speed oscillator (**fiL**)
- External 32KHz Oscillator (Sub Clock) (**fsub**)
- External signal input (**TI05**)

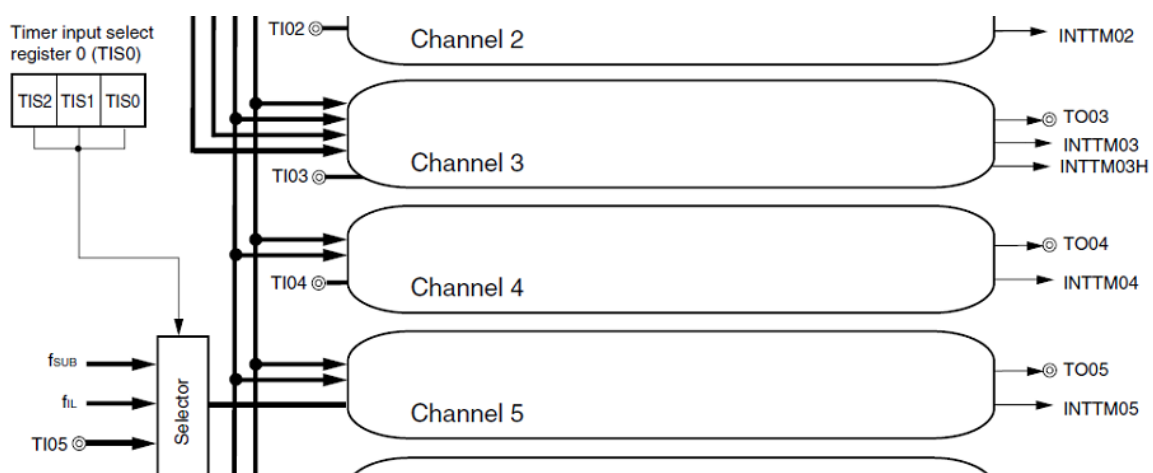


Figure 8 Timer Array Unit Channel 5 Configuration

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. This is a more accurate method of measurement than the software approach.

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 = 32KHz

Therefore the calculation is simply  $32000000 / 32768 = 976$  (h'3D0)

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

### 2.4.2. Software Measurement

The principle behind the software measurement is based on a software counter measuring the transition on the test port pin. The actual comparison values can be a mix of calculation and measurement as it is difficult to fully calculate the measurement value due to variances in the synchronisation and monitoring of the input state.

The measurement sequence is

- Synchronise to the reference clock (high to low transition on the input pin)
- Wait for the next low to high transition and then start the software counter
- Increment the software count until the next high to low transition
- Compare the software count value against the high and lower limit reference values

The basic calculation is based on the following equation

$$\text{System Clock} / (\text{Reference Clock} / 2) \times \text{the number of clock cycles executed in the count loop}$$

Note: The measurement period of the software counter is based on half the reference clock

Using the example settings provided in the test harness project

The System clock is 32MHz and the reference clock is the Sub Clock 32KHz then the calculation is

$$32000000 / (32768 / 2) \times \text{Loop Count}$$

The cycle count can be calculated as shown in the code extract in figure 9 below

$$\frac{1}{2} \text{ the reference clock} = 15.26\mu\text{S} (32\text{KHz} / 2)$$

The loop count of the measurement period ([measure high time](#)) is 6 clock cycles

$$\text{At 32MHz this is } 187.5\text{nS} (6 \times 31.25\text{nS})$$

Therefore the approximate software count for the test harness example is  $15.26\mu\text{S} / 187\text{nS} = 82$  (h'52)

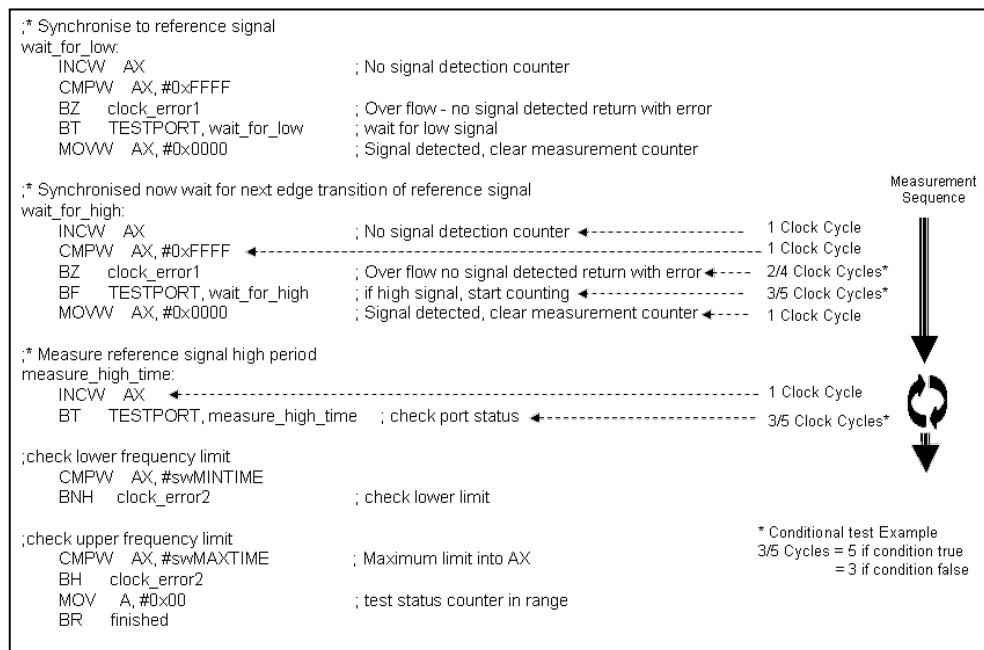


Figure 9 Timer Array Unit Channel 5 Configuration

Table 15: Source files: Software Clock test

STL File name	Header Files
<a href="#">stl_RL78_sw_clocktest.asm</a>	<a href="#">stl_clocktest.h</a> <a href="#">stl.h</a>
Test Harness File Names	Header Files
<a href="#">main.c</a> <a href="#">stl_global_data_example.c</a> <a href="#">stl_main_example_support function.c</a> <a href="#">stl_peripheralinit.c</a>	<a href="#">main.h</a> <a href="#">stl_gobal_data_example.h</a>

Declaration	
char stl_RL78_sw_clocktest(void)	
Description	
<p>This function tests the system clock using a software measurement (software counter) process. The measured result (software count) is compared against the upper and lower limit values defined in the clock test header file (<a href="#">stl_clocktest.h</a>), and the result status (Pass / Fail / No reference clock) is returned to the calling function.</p> <p>The reference limits calculation is based on the following</p> $\text{System Clock} / (\text{Reference Clock} / 2) \times \text{times the number of clock cycles executed in the count loop}$ <p>The function “<a href="#">indicate_test_result</a>” will be called by the test harness control files (<a href="#">main.c</a>) to process the test result  Note Function “<a href="#">indicate_test_result</a>” is located in the module <a href="#">stl_main_example_support function.c</a></p>	
Input Parameters	
swMAXTIME	Upper time limit compare value (Defined in <a href="#">stl_clocktest.h</a> )
swMINTIME	Lower time limit compare value (Defined in <a href="#">stl_clocktest.h</a> )
TESTPORT	Test Port Input Pin for external reference signal input (Defined in <a href="#">stl_clocktest.h</a> )
Output Parameters	
NONE	N/A
Return Values	
char	<p>Test result of CPU register C</p> <p>0 = Test passed.</p> <p>1 = Test measurement failed (Outside the reference window)</p> <p>2 = Test measurement failed (No reference clock detected)</p>

Table 16: Source files: Hardware Clock test

STL File name	Header Files
<a href="#">stl_RL78_hw_clocktest.asm</a>	<a href="#">stl_clocktest.h</a> <a href="#">stl.h</a>
Test Harness File Names	Header Files
<a href="#">main.c</a> <a href="#">stl_global_data_example.c</a> <a href="#">stl_main_example_support function.c</a> <a href="#">stl_peripheralinit.c</a>	<a href="#">main.h</a> <a href="#">stl_gobal_data_example.h</a>

Declaration	
char stl_RL78_hw_clocktest(void)	
Description	
<p>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 (<a href="#">stl_clocktest.h</a>) and the result status (Pass / Fail / No reference clock) is returned to the calling function.</p> <p>The function “<a href="#">indicate_test_result</a>” will be called by the test harness control files (<a href="#">main.c</a>) to process the test result. Note Function “<a href="#">indicate_test_result</a>” is located in the module <a href="#">stl_main_example_support function.c</a></p>	
Input Parameters	
hwMAXTIME	Upper time limit compare value (Defined in <a href="#">stl_clocktest.h</a> )
hwMINTIME	Lower time limit compare value (Defined in <a href="#">stl_clocktest.h</a> )
CAPTURE_interrupt_FLAG	Timer channel Capture Interrupt Flag (Defined in <a href="#">stl_clocktest.h</a> )
Output Parameters	
NONE	N/A
Return Values	
Char	<p>Test result of CPU register C</p> <p>0 = Test passed.</p> <p>1 = Test measurement failed ( Outside the reference window)</p> <p>2 = Test measurement failed (No reference clock detected)</p>

### 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-Up 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 Ram using Initial March C (or initial March X)

All register tests

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 tests.            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 Tests.      These are dependant 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 Verification

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- Up 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 are 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 Verification

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 + tool chain can be used to calculate and add a CRC value and place at a location specified by the user. CS + grants three types of CRC: “general-purpose CRC”, “high-speed CRC (CCR-16-CCITT)”, and “high-speed CRC (SENT)”. Hardware CRC calculation provide in this library (function “stl\_RL78\_peripheral”) and the C-language function to generate the reference CRC value (function “reference\_crc\_calculation”) correspond to the CS+’s “general-purpose CRC”.

The reference value of the software CRC adopted in this library (function “stl\_RL78\_sw\_crc\_asm”) can NOT be generated by CS +. It has to be made by the users themselves referring to the algorithm shown in the source files.

See Figure 16: CS + object convert option

The reference CRC value of the software CRC (stl\_RL78\_sw\_crc\_asm function) implemented in this library can not be generated with CS +. Please refer to reference\_crc\_CCIT16\_Msb\_calculation.

See Figure 10: Adding Reference CRC.

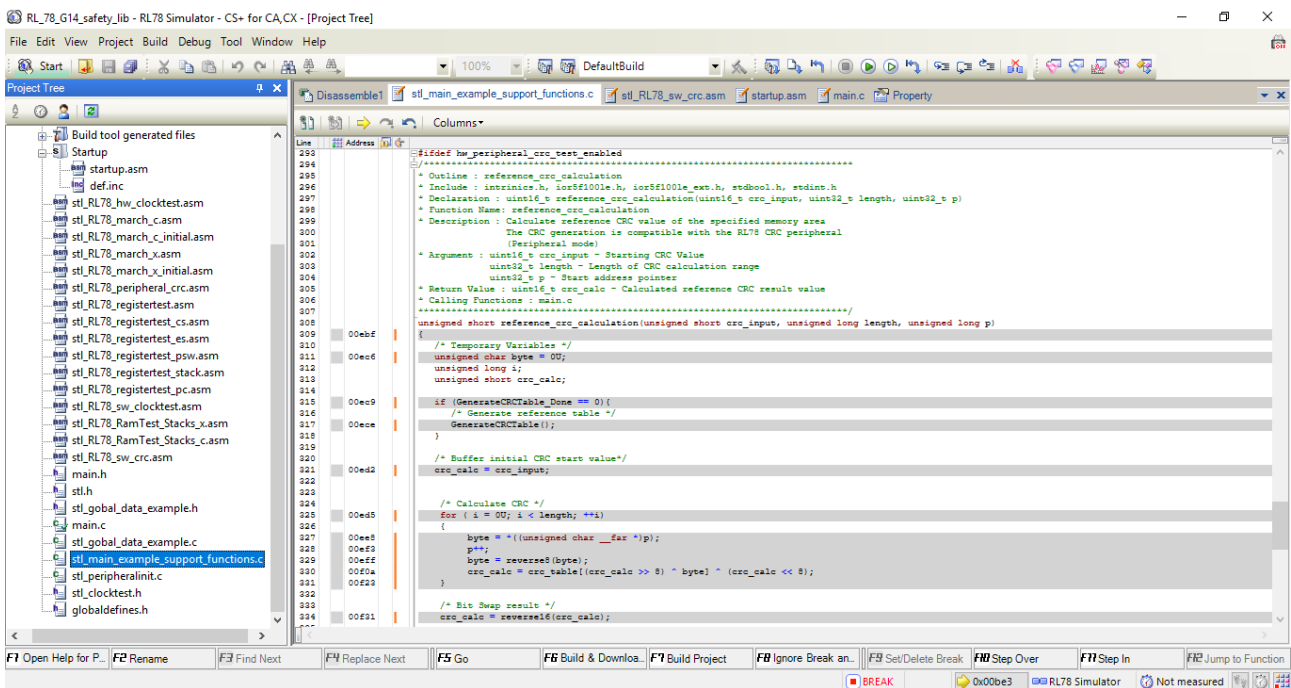


Figure 10: Adding Reference CRC

### **3.2.1. Power- Up 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 Verification**

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

1. RAM being tested can not be used for anything else including the current stack.
2. Any test requires a RAM buffer where memory contents can be safely copied to and restored from.
3. 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 can not be performed during this operation.

### **3.3.1. Power-Up**

It is recommended to use the “initial RAM test modules (march C or March X), as these are specifically design for testing all of the Ram area at power on or Reset. The modules have been designed without any function call and so are suitable to be executed before the system and C-Stack are initialised as any contents of the Ram memory will be destroyed. In this library, those initial RAM test modules are implemented in 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 Verification

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 system can be controlled using a different known clock.

#### 3.4.1. Power-Up

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

#### 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.

(i.e. System clock = 32 MHZ, Reference clock = 15 KHz)

### 3.5. Code Coverage

The code coverage can be checked by observing the function list section in simulator mode.

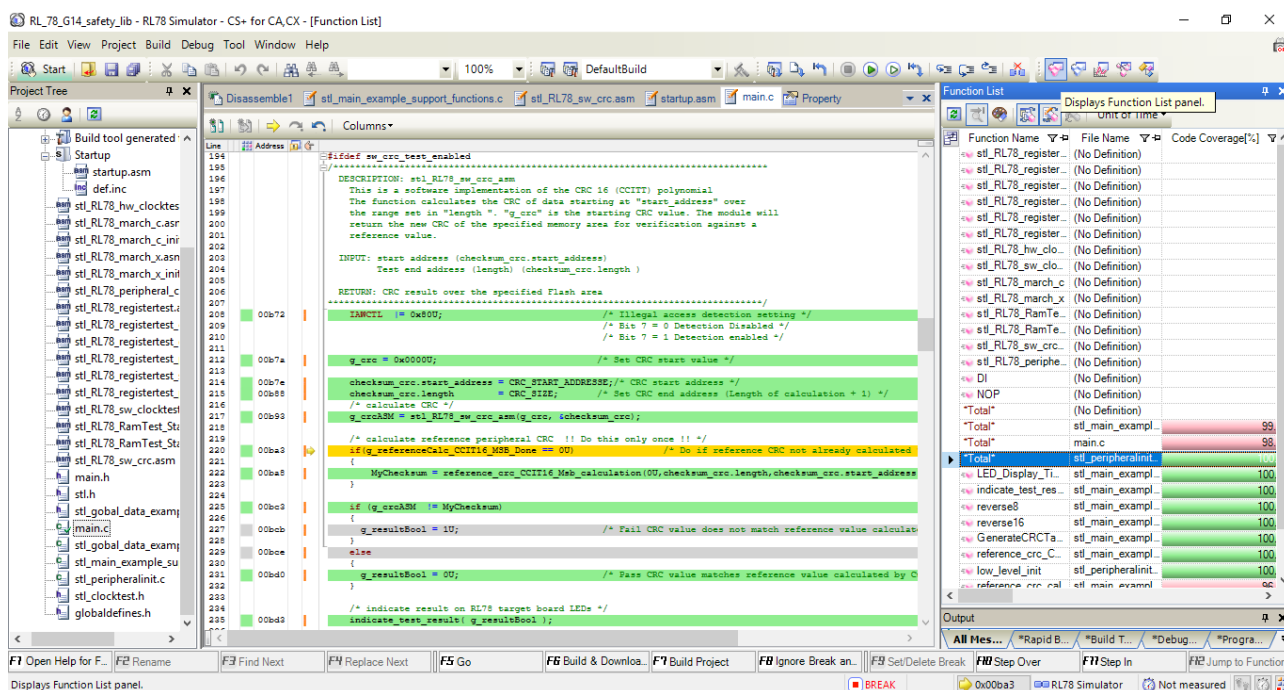


Figure 11 Function list section

## 4. Benchmarking

### 4.1. Development Environment

- E1(R0E000010KCE00) On-chip debugging emulator
- QB-R5F104PJ-TB RL78/G14 Target Board (100pin LQFP , 14 x 14mm)
- Tool chain: CS+ for CA/CX V4.02.00 CA78K0R V1.72

MCU: R5F104PJAFB

Internal Clock: 32 MHz High Speed Oscillator  
System Clock = 32 MHz

External Sub Clock: 32 KHz

### 4.2. 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+.

#### 4.2.1. General Options

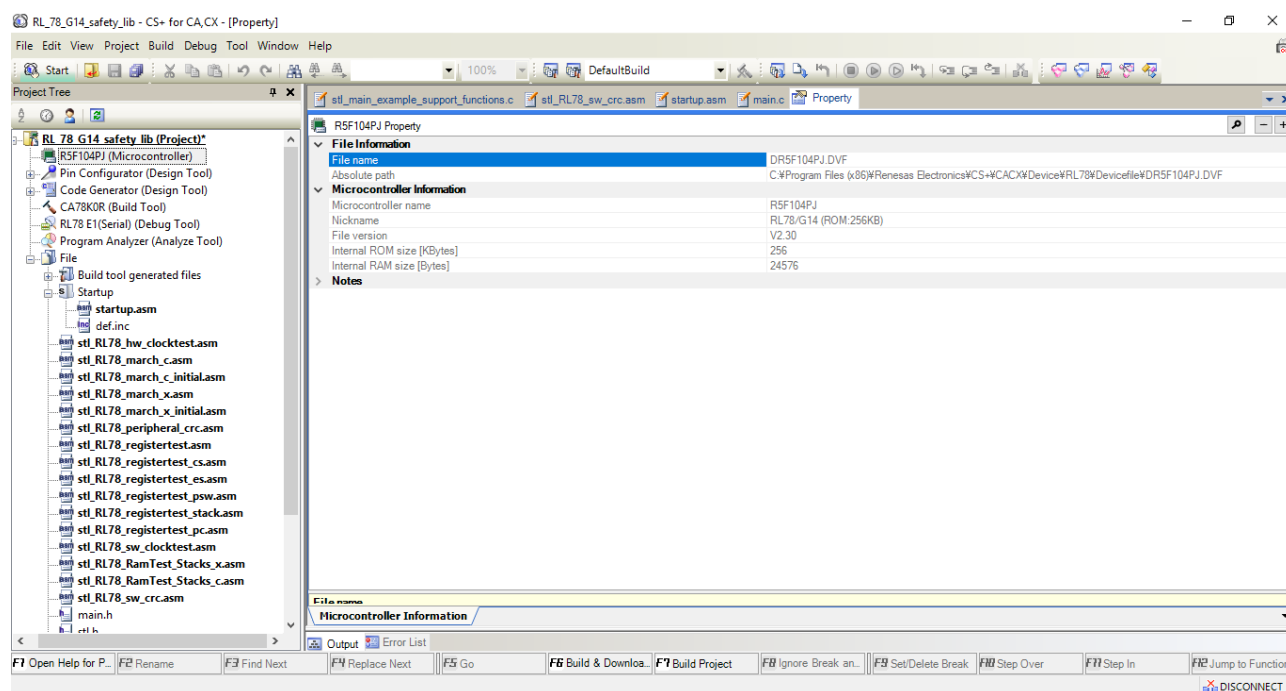


Figure 12 CS+ common Options - Target Device

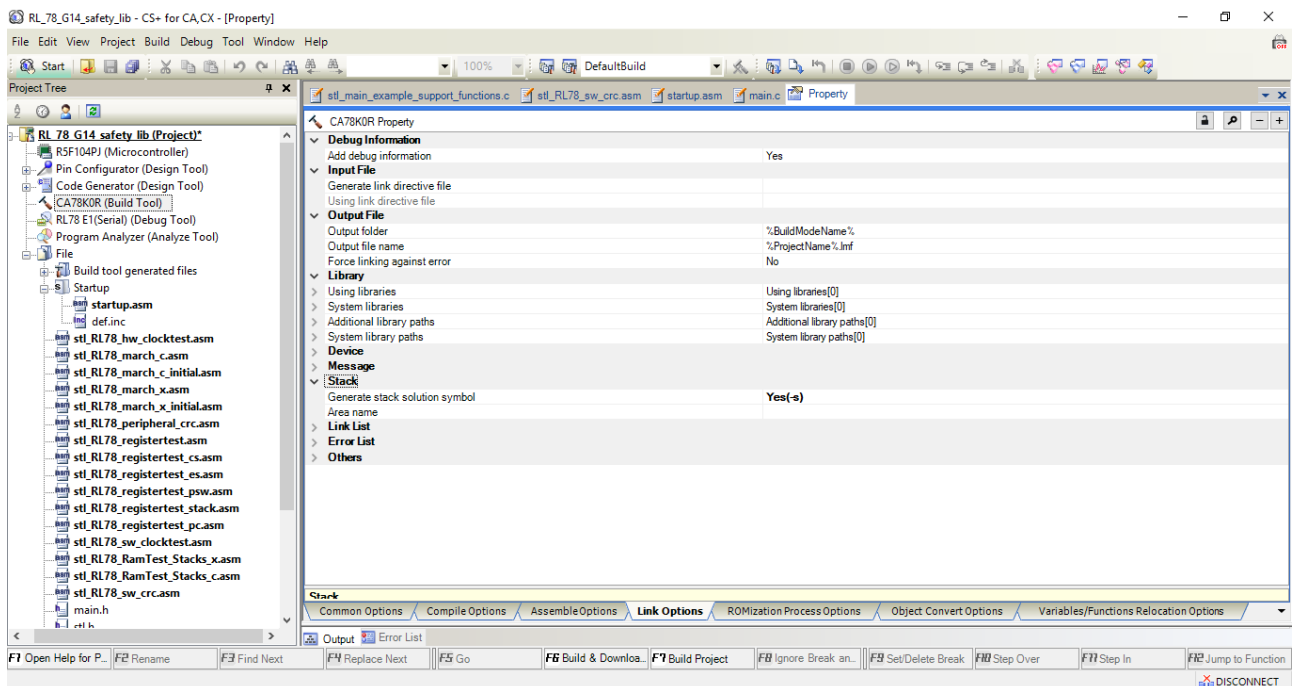


Figure 13 CS+ Link Options

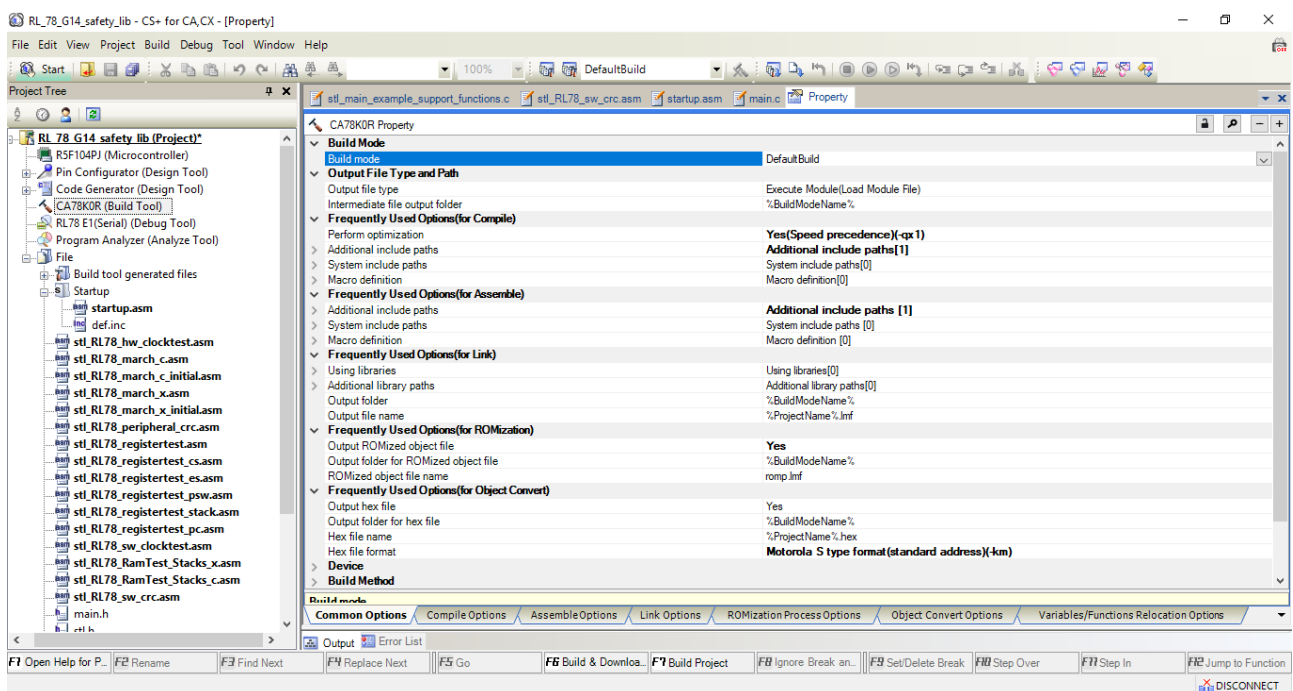


Figure 14 CS+ Common Options

## 4.2.2. Compiler Settings

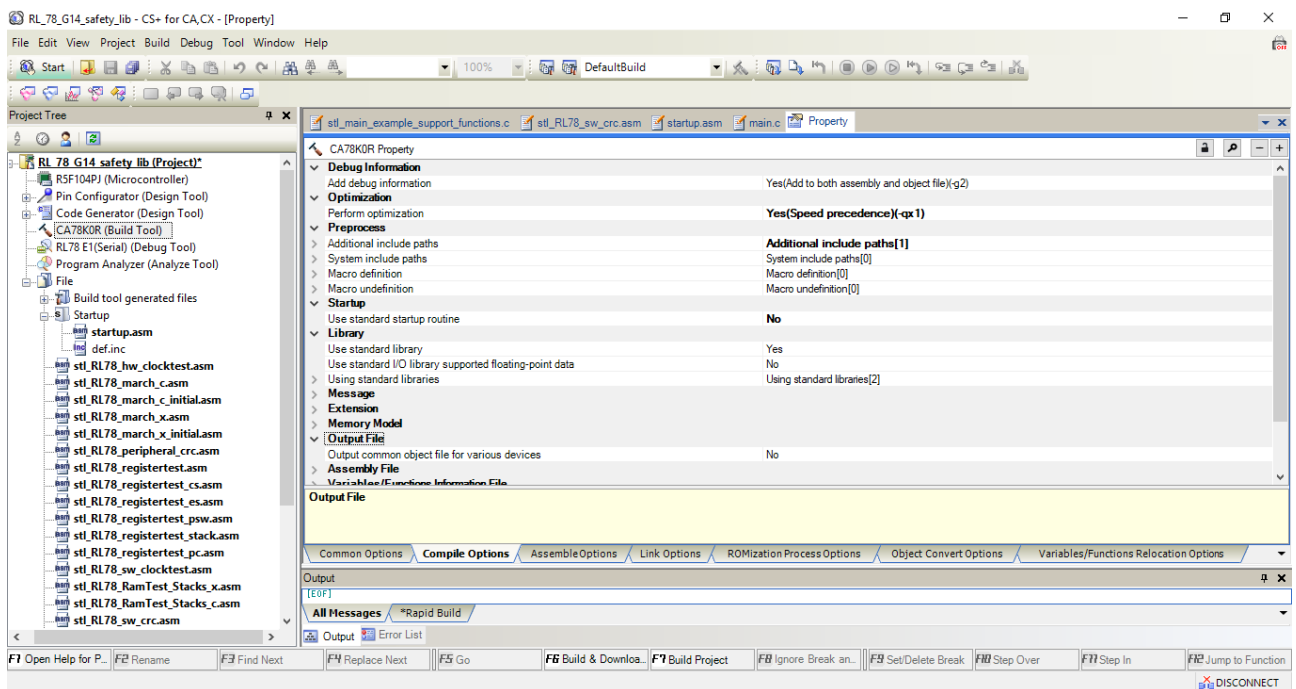


Figure 15 CS+ Compiler Options

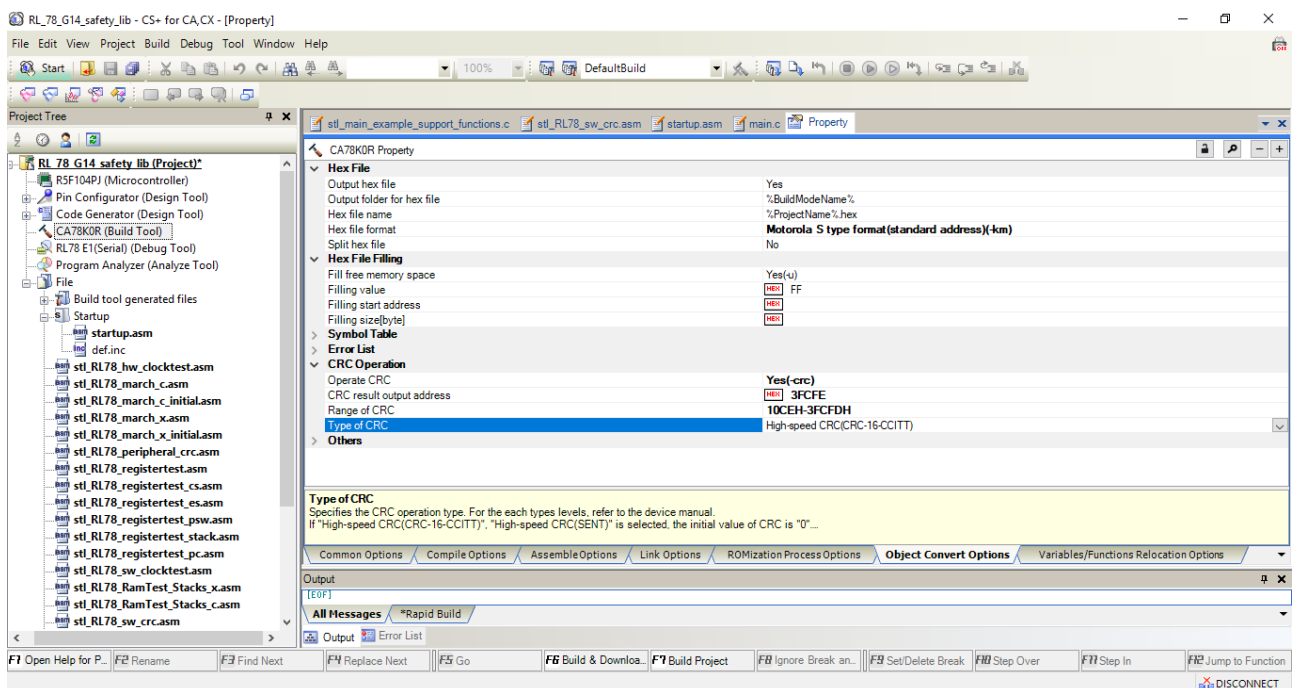


Figure 16 CS + Object Convert Options

## 4.3. Benchmark test results

Library functions	Number of bytes tested	Processing time
<b>CPU Register Tests - Software API</b> stl_RL78_registertest	-	10.312μs
<b>CPU Registers Tests – PSW</b> stl_RL78_registertest_psw	-	1.375μs
<b>CPU Registers Tests - SP</b> stl_RL78_registertest_stack	-	1.156μs
<b>CPU Registers Tests - CS</b> stl_RL78_registertest_cs	-	1.062μs
<b>CPU Registers Tests - ES</b> stl_RL78_registertest_es	-	1.062μs
<b>CPU Registers Tests - PC</b> stl_RL78_registertest_pc	-	2.218μs
<b>Software CRC</b> stl_RL78_sw_crc_asm	257072byte	394200μs(394.200ms)
<b>Hardware CRC</b> stl_RL78_peripheral_crc	257072byte	185000μs (185.000ms)
<b>System March C</b> stl_RL78_march_c	128byte	1434μs(1.434ms)
<b>System March X</b> stl_RL78_march_x	128byte	797.906μs
<b>Initial March C</b> stl_RL78_march_c_initia	24540byte	274542μs (274.542ms)
<b>Initial March X</b> stl_RL78_march_x_initial	24540byte	152609μs(152.609ms)
<b>Hardware Clock test</b> stl_RL78_hw_clocktest	-	56.40μs
<b>Software Clock test</b> stl_RL78_sw_clocktest	-	5700μs (5.700ms)
<b>Stack area test (March C)</b> stl_RL78_RamTest_Stacks_c	512byte+512byte	11782μs (11.782ms)
<b>Stack area test (March X)</b> stl_RL78_RamTest_Stacks_x	512byte+512byte	6694μs(6.694ms)

## 5. 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.

### 5.1. Additional Safety Functions

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

#### 5.1.1. RAM Memory 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.

Please note that this function is available only for data accesses and does not apply to code executed from Ram.

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 After 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	Enable parity error resets.
1	Disable parity error resets.

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

Figure 17 RAM Parity Error Checking



### 5.1.2. 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 22 below:

Format of Invalid Memory Access Detection Control Register (IAWCTL)

Address: F0078H After reset: 00H R/W

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

IAWEN <sup>Note 1</sup>	Control of invalid memory access detection
0	Disable the detection of invalid memory access.
1	Enable the detection of invalid memory access.

GRAM1	GRAM0	RAM guard space <sup>Note 2</sup>
0	0	Disabled. RAM can be written to.
0	1	The 128 bytes starting at the lower RAM address
1	0	The 256 bytes starting at the lower RAM address
1	1	The 512 bytes starting at the lower RAM address

Figure 18 RAM Guard Protection

### 5.1.3. 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 After reset: 00H R/W

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

IAWEN <sup>Note 1</sup>	Control of invalid memory access detection
0	Disable the detection of invalid memory access.
1	Enable the detection of invalid memory access.

Figure 19 Invalid Memory Access Protection

#### 5.1.4. 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, PMCxx, ADPC, and PIOR  
Pxx cannot be guarded.

The Port I/O SFR registers can be guarded by the “**GPORT**” bit as shown in figure 20 below

Format of Invalid Memory Access Detection Control Register (IAWCTL)

Address: F0078H    After reset: 00H    R/W								
Symbol	7	6	5	4	3	2	1	0
IAWCTL	IAWEN	0	GRAM1	GRAM0	0	GPORT	GINT	GCSC

IAWEN <sup>Note 1</sup>	Control of invalid memory access detection	
0	Disable the detection of invalid memory access.	
1	Enable the detection of invalid memory access.	

GRAM1	GRAM0	RAM guard space <sup>Note 2</sup>
0	0	Disabled. RAM can be written to.
0	1	The 128 bytes starting at the lower RAM address
1	0	The 256 bytes starting at the lower RAM address
1	1	The 512 bytes starting at the lower RAM address

GPORT	Port register guard <sup>Note 3</sup>
0	Disabled. Port registers can be read or written to.
1	Enabled. Writing to port registers is disabled. Reading is enabled.

Figure 20 I/O Port SFR Guard Protection

### 5.1.5. 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 21 below

Format of Invalid Memory Access Detection Control Register (IAWCTL)

Address: F0078H After reset: 00H R/W

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

IAWEN <sup>Note 1</sup>	Control of invalid memory access detection
0	Disable the detection of invalid memory access.
1	Enable the detection of invalid memory access.

GRAM1	GRAM0	RAM guard space <sup>Note 2</sup>
0	0	Disabled. RAM can be written to.
0	1	The 128 bytes starting at the lower RAM address
1	0	The 256 bytes starting at the lower RAM address
1	1	The 512 bytes starting at the lower RAM address

GPORT	Port register guard <sup>Note 3</sup>
0	Disabled. Port registers can be read or written to.
1	Enabled. Writing to port registers is disabled. Reading is enabled.

GINT	Interrupt register guard <sup>Note 4</sup>
0	Disabled. Interrupt registers can be read or written to.
1	Enabled. Writing to interrupt registers is disabled. Reading is enabled.

Figure 21 Interrupt SFR Guard Protection

### 5.1.6. 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, and RPECTL

The interrupt SFR registers can be guarded by the “GCSC” bit as shown in figure 22 below

Format of Invalid Memory Access Detection Control Register (IAWCTL)

Address: F0078H    After reset: 00H    R/W

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

IAWEN <sup>Note 1</sup>	Control of invalid memory access detection
0	Disable the detection of invalid memory access.
1	Enable the detection of invalid memory access.

GRAM1	GRAM0	RAM guard space <sup>Note 2</sup>
0	0	Disabled. RAM can be written to.
0	1	The 128 bytes starting at the lower RAM address
1	0	The 256 bytes starting at the lower RAM address
1	1	The 512 bytes starting at the lower RAM address

GPORT	Port register guard <sup>Note 3</sup>
0	Disabled. Port registers can be read or written to.
1	Enabled. Writing to port registers is disabled. Reading is enabled.

GINT	Interrupt register guard <sup>Note 4</sup>
0	Disabled. Interrupt registers can be read or written to.
1	Enabled. Writing to interrupt registers is disabled. Reading is enabled.

GCSC	Chip state control register guard <sup>Notes 5, 6</sup>
0	Disabled. Chip state control registers can be read or written to.
1	Enabled. Writing to chip state control registers is disabled. Reading is enabled.

**Figure 22 Invalid Memory Access Protection**

## 5.2. Additional Self Test Functions

The ADC includes additional inputs designed to help test the operation of the ADC. These include

- Temperature Sensor
- Internal Voltage Reference (1.44V)
- External Analogue Voltage Reference pins (AVrefP and AVrefM)

These internal analogue input pins can be used to verify the operation of the ADC against a known reference point. The external pins can be set to (typically AVrefP < Vdd, AVrefM = Vss) additional measurement point to establish the correct operation of the ADC.

Note the normal ADC input selection register (ADS) can be used for all inputs except the external reference inputs which are set according to the table below.

Address: F0013H	After reset: 00H	R/W							
Symbol	7	6	5	4	3	2	1	0	
ADTES	0	0	0	0	0	ADTES2	ADTES1	ADTES0	

ADTES2	ADTES1	ADTES0	A/D conversion target
0	0	0	AN <sub>xx</sub> (This is specified using the analog input channel specification register (ADS).)
0	1	0	AV <sub>REFM</sub>
0	1	1	AV <sub>REFP</sub>
Other than the above			Setting prohibited

**Figure 23 ADC Self Test**

### 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(R01AN0749E)

Website and Support

Renesas Electronics Website

<http://www.renesas.com/>

Inquiries

<http://www.renesas.com/inquiry>

All trademarks and registered trademarks are the property of their respective owners.

## Revision Record

[illegible]

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

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

## 1. Precaution against Electrostatic Discharge (ESD)

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

## 2. Processing at power-on

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

## 3. Input of signal during power-off state

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

## 4. Handling of unused pins

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

## 5. Clock signals

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

## 6. Voltage application waveform at input pin

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

## 7. Prohibition of access to reserved addresses

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

## 8. Differences between products

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



## Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

"Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.

"High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user's manual or other Renesas Electronics document.

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

(Note1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

## Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,  
Koto-ku, Tokyo 135-0061, Japan  
[www.renesas.com](http://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/](http://www.renesas.com/contact/).