RL78 Family

VDE Certified IEC60730/60335 Self Test Library

R01AN0749EG0201
Rev.2.01
Mar 04, 2014

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 routines have been certified by VDE Test and Certification Institute GmbH. A copy of the Test Certificate is available in the download package for this Application Note together with the certified self test library source code and the test harness IAR project

Although these routines were developed using IEC60730/60335 compliance as a basis, they can be implemented in any system for self testing of Renesas Microcontroller families.

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 Family

Contents

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 IAR Embedded Workbench test harness allows each of the tests functions to be selected in turn and run as a stand alone function. In order to minimise the affects of the optimisation in the C compiler and minimise resources used, all of the self test library files have been written in assembler. The default build of the test harness C files has been built with the optimisation set to "None" in the IAR Embedded workbench.

All of the STL modules and test harness files are MISRA-C compliant

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.

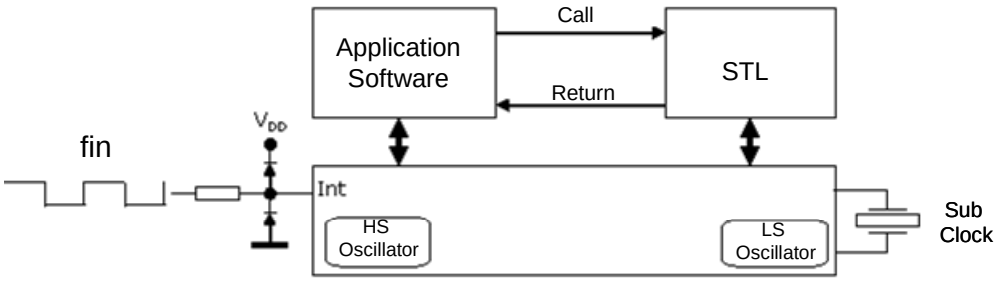

Figure 1 Self Test Library (STL) Configuration

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.
  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.11.12.7

- Invariable Memory

  This tests the MCU internal Flash memory
  IEC Reference - IEC 60730: 1999+A1:2003 Annex H - Table H.11.12.7

- Variable Memory

  This tests the Internal SRAM memory
  IEC Reference - IEC 60730: 1999+A1:2003 Annex H - Table H.11.12.7

- System Clock

  Verifies the system clock operation and correct frequency against a reference clock source
  IEC Reference - IEC 60730: 1999+A1:2003 Annex H - Table H.11.12.7

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

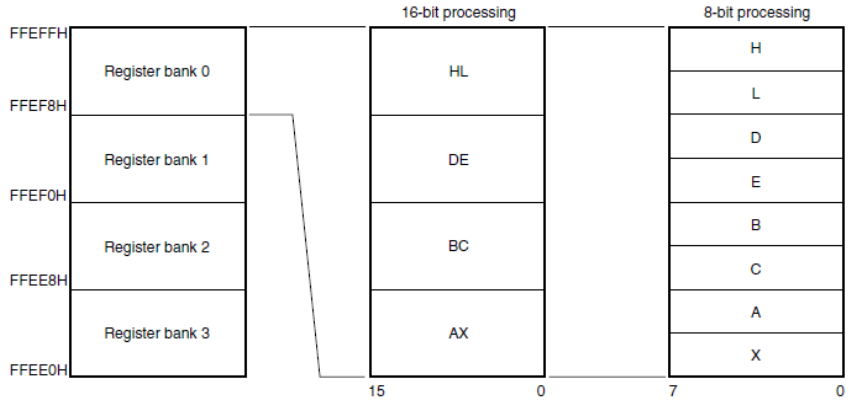

Figure 2 Working Register Configuration

- **Stack Pointer** (SP)

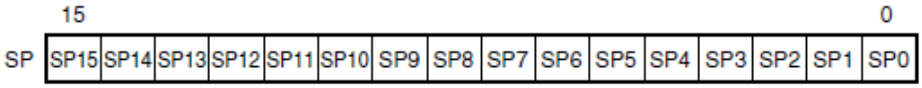

Figure 3 Stack Pointer Configuration

- **Processor Status Word** (PSW)

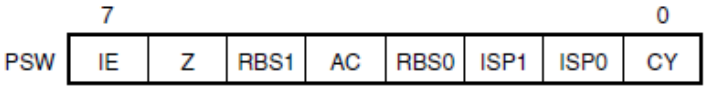

Figure 4 PSW Register Configuration

- **Code Address Extension Register** (CS)

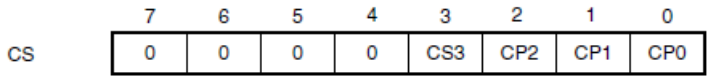

Figure 5 Working Register Configuration

- **Data Address Extension Register** (ES)

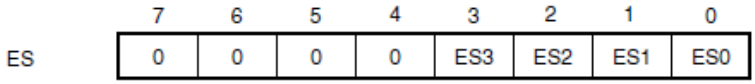

Figure 6 Working Register Configuration

2.1.1 CPU Register Tests - Software API

Table 1: Source files: CPU Working Registers Tests

STL File name	Header Files
stl_RL78_registertest.asm	None
Test Harness File Names	**Header Files**
main.c stl_global_data_example.c stl_main_example_support function.c stl_peripheralinit.c	stl.h main.h stl_gobal_data_example.h

Syntax	
bool stl_RL78_registertest(void)	
Description	
This module tests the RL78 working registers and accumulators. Registers AX, HL, DE, BC in all three register banks (Banks 0, 1, 2, 3) These registers are tested as16bit registers. The following tests are performed for each register: 1. Write h'5555 to the register being tested. 2. Read back and check they are equal. 3. Write h'AAAA to the register being tested. 4. Read back and check they are equal. It is the calling function's responsibility to ensure no interrupts occur during this test. The original register contents are restored on completion of the test The function "indicate_test_result.c" will be called by the test harness control files (main.c) to process the test result Note Function "indicate_test_result.c "is located in the module stl_main_example_support function.c	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
bool	Test Result Status returned in CPU Register A 0 = Test passed. 1 = Test or parameter check failed

Table 2: Source files: CPU Registers Tests – PSW

STL File name	Header Files
stl_RL78_registertest_psw.asm	stl.h
Test Harness File Names	**Header Files**
main.c stl_global_data_example.c stl_main_example_support function.c stl_peripheralinit.c	main.h stl_gobal_data_example.h

Syntax	
bool stl_RL78_registertest_psw(void)	
Description	
Test the 8bit Processor Status Word (PSW) register The following tests are performed: 1. Write h'55 to the register being tested. 2. Read back and check it is equal. 3. Write h'AA to the register being tested. 4. Read back and check that it is equal. 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 "indicate_test_result.c" will be called by the test harness control files (main.c) to process the test result Note Function "indicate_test_result.c "is located in the module stl_main_example_support function.c	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
bool	Test Result Status returned in CPU Register A 0 = Test passed. 1 = Test or parameter check failed

Table 3: Source files: CPU Registers Tests - SP

STL File name	Header Files
stl_RL78_registertest_stack.asm	stl.h
Test Harness File Names	**Header Files**
main.c stl_global_data_example.c stl_main_example_support function.c stl_peripheralinit.c	main.h stl_gobal_data_example.h

Syntax	
bool stl_RL78_registertest_stack(void)	
Description	
Test the 16bit Stack Pointer (SP) register The following tests are performed: 1. Write h'5555 to the register being tested. 2. Read back and check it is equal. 3. Write h'AAAA to the register being tested. 4. Read back and check that it is equal. 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 "indicate_test_result.c" will be called by the test harness control files (main.c) to process the test result Note Function "indicate_test_result.c "is located in the module stl_main_example_support function.c	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
bool	Test Result Status returned in CPU Register A 0 = Test passed. 1 = Test or parameter check failed

Table 4: Source files: CPU Registers Tests - CS

STL File name	Header Files
stl_RL78_registertest_cs.asm	stl.h
Test Harness File Names	**Header Files**
main.c stl_global_data_example.c stl_main_example_support function.c stl_peripheralinit.c	main.h stl_gobal_data_example.h

Syntax	
bool stl_RL78_registertest_cs(void)	
Description	
Test the 8bit code extension (CS) register The following tests are performed: 1. Write h'05 to the register being tested. 2. Read back and check it is equal. 3. Write h'0A to the register being tested. 4. Read back and check that it is equal. Please note that the top 4 bit 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 "indicate_test_result.c" will be called by the test harness control files (main.c) to process the test result Note Function "indicate_test_result.c "is located in the module stl_main_example_support function.c	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
bool	Test Result Status returned in CPU Register A 0 = Test passed. 1 = Test or parameter check failed

Table 5: Source files: CPU Registers Tests - ES

STL File name	Header Files
stl_RL78_registertest_es.asm	stl.h
Test Harness File Names	**Header Files**
main.c stl_global_data_example.c stl_main_example_support function.c stl_peripheralinit.c	main.h stl_gobal_data_example.h

Syntax	
bool stl_RL78_registertest_es(void)	
Description	
Test the 8bit data extension (ES) register The following tests are performed: 1. Write h'05 to the register being tested. 2. Read back and check it is equal. 3. Write h'0A to the register being tested. 4. Read back and check that it is equal. Please note that the top 4 bit 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 "indicate_test_result.c" will be called by the test harness control files (main.c) to process the test result Note Function "indicate_test_result.c "is located in the module stl_main_example_support function.c	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
bool	Test Result Status returned in CPU Register A 0 = Test passed. 1 = Test or parameter check failed

2.2 Invariable Memory Test – Flash ROM

This section describes the Flash memory test using CRC routines. CRC is a fault / error control technique which generates a single word or checksum to represent the contents of memory. A CRC checksum is the remainder of a binary division with no bit carry (XOR used instead of subtraction), of the message bit stream, by a predefined (short) bit stream of length n + 1, which represents the coefficients of a polynomial with degree n. Before the division "n" zeros are appended to the message stream. CRCs are popular because they are simple to implement in binary hardware and are easy to analyse mathematically.

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

The 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 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 software implementation will produce the same result as the IAR Embedded workbench Standard tool chain using the "extra options" as provided in the Linker configuration menu for CRC test harness projects. Therefore if you are using the IAR tool chain to automatically insert a reference CRC into the ROM the value can be compared directly with the one calculated.

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 6: Source files: Software CRC

STL File name	Header Files
stl_RL78_sw_crc.asm	stl.h
Test Harness File Names	**Header Files**
main.c stl_global_data_example.c stl_main_example_support function.c stl_peripheralinit.c	main.h stl_gobal_data_example.h

Syntax	
uint16_t stl_RL78_sw_crc_asm (uint16_t crc, CHECKSUM_CRC_TEST_AREA *p);	
Description	
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. The function "indicate_test_result.c" will be called by the test harness control files (main.c) to process the test result Note Function "indicate_test_result.c "is located in the module stl_main_example_support function.c	
Input Parameters	
uint16_t 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	
uint16_t	16 bit calculated CRC value (Full or partial result) is returned in CPU Register AX

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 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 (stl_RL78_sw_crc.asm) by the calling function in main.c	
Input Parameters	
uint32_t length;	Range (length = number of bytes +1) of memory to be tested.
uint32_t 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 7: Source files: Hardware CRC Calculation

STL File name	**Header Files**
stl_RL78_peripheral_crc.asm	<ior5f100le.h> <ior5f100le_ext.h> stl.h
Test Harness File Names	**Header Files**
main.c stl_global_data_example.c stl_main_example_support function.c stl_peripheralinit.c	main.h stl_gobal_data_example.h

Syntax	
uint16_t stl_RL78_peripheral_crc(uint16_t gcrc, CHECKSUM_CRC_TEST_AREA *p)	
Description	
This function calculates a CRC value over the address range supplied using the hardware CRC peripheral. 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 function "indicate_test_result.c" will be called by the test harness control files (main.c) to process the test result Note Function "indicate_test_result.c "is located in the module stl_main_example_support function.c	
Input Parameters	
uint16_t gcrc	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	
uint16_t	16 bit calculated CRC value (Full or partial result) is returned in CPU Register AX

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 (stl_RL78_peripheral_crc.asm) by the calling function in main.c. Note: This is the same structure as used by the software CRC function.	
Input Parameters	
uint32_t length;	Range (length = number of bytes + 1) of memory to be tested.
uint32_t start_address	Start address for CRC calculation
Output Parameters	
NONE	N/A
Return Values	
	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 design 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.

The area of RAM being tested can not be used during testing, 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 (<>(w0))
2. Starting at lowest address, read zeros, write ones, increment up array bit by bit. (>(r0,w1))
3. Starting at highest address, read ones, write zeros, decrement down array bit by bit. (<(r1,w0))
4. Read all zeros from array. (<>(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.

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 8: Source files: System March C

STL File name	Header Files
stl_RL78_march_c.asm	stl.h
Test Harness File Names	**Header Files**
main.c stl_global_data_example.c stl_main_example_support function.c stl_peripheralinit.c	main.h stl_gobal_data_example.h

Declaration	
bool stl_RL78_march_c(uint16_t num, uint8_t *addr)	
Description	
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. The function "indicate_test_result.c" will be called by the test harness control files (main.c) to process the test result Note Function "indicate_test_result.c "is located in the module stl_main_example_support function.c	
Input Parameters	
uint8_t *addr	Pointer to the start address of the RAM to be tested.
uint16_t num	The range (Number of bytes +1) of the RAM to be tested.
Output Parameters	
NONE	N/A
Return Values	
bool	Test status result is returned in CPU register A 0 = Test passed. 1 = Test or parameter check failed

2.3.2.2 System March X

The system March X self test function is the essentially the same as the system March C module except that it only implements the reduced March X algorithm. The module is designed to run after the application system has been initialised and so should not be used to test the whole memory area in a single operation.

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:

STL File name	**Header Files**
stl_RL78_march_x.asm	stl.h
Test Harness File Names	**Header Files**
main.c stl_global_data_example.c stl_main_example_support function.c stl_peripheralinit.c	main.h stl_gobal_data_example.h

Declaration	
bool stl_RL78_march_c(uint16_t num, uint8_t *addr)	
Description	
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. The function "indicate_test_result.c" will be called by the test harness control files (main.c) to process the test result Note Function "indicate_test_result.c "is located in the module stl_main_example_support function.c	
Input Parameters	
uint8_t *addr	Pointer to the start address of the RAM to be tested.
uint16_t num	The range (Number of bytes +1) of the RAM to be tested.
Output Parameters	
NONE	N/A
Return Values	
bool	Test status result is returned in CPU register A 0 = Test passed. 1 = Test or parameter check failed

2.3.2.3 Initial March C

The initial March C test is designed to run before the application system has been 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 "cstartup.s87" module and return to "cstartup.s87" 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 10: Source files:

STL File name	**Header Files**
stl_RL78_march_c_initial.asm	None
Test Harness File Names	**Header Files**
ctsartup.s87	None

Declaration	
stl_RL78_march_c_initial	
Description	
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. The function "indicate_test_result.c" will be called by the test harness control files (main.c) to process the test result Note Function "indicate_test_result.c "is located in the module stl_main_example_support function.c	
Input Parameters	
CPU Register AX	16bit Register holding the start address of the RAM to be tested.
CPU Register BC	16bit Register holding the range (Number of bytes + 1) of the RAM to be tested.
Output Parameters	
NONE	N/A
Return Values	
CPU Register A	Test status result 0 = Test passed. 1 = Test or parameter check failed

2.3.2.4 Initial March X

The initial March X test is designed to run before the application system has been 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 "cstartup.s87" module and return to "cstartup.s87" 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:

STL File name	**Header Files**
stl_RL78_march_x_initial.asm	None
Test Harness File Names	**Header Files**
ctsartup.s87	None

Declaration	
stl_RL78_march_x_initial	
Description	
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. The function "indicate_test_result.c" will be called by the test harness control files (main.c) to process the test result Note Function "indicate_test_result.c "is located in the module stl_main_example_support function.c	
Input Parameters	
CPU Register AX	16bit Register holding the start address of the RAM to be tested.
CPU Register BC	16bit Register holding the range (Number of bytes + 1) of the RAM to be tested.
Output Parameters	
NONE	N/A
Return Values	
CPU Register A	Test status result in returned in register "A" 0 = Test passed. 1 = Test or parameter check failed.

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.

1. Hardware Measurement

All current RL78 devices include an option in the Timer Array Unit (TAU) that provides additional input capture sources that are designed to be able 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 (TIOn)

The example shown below is for the R5F100LE device, which has this feature implemented on TAU channel 5. The timer channel used for other RL78 family members can vary, however the principles of this test capability remain the same.

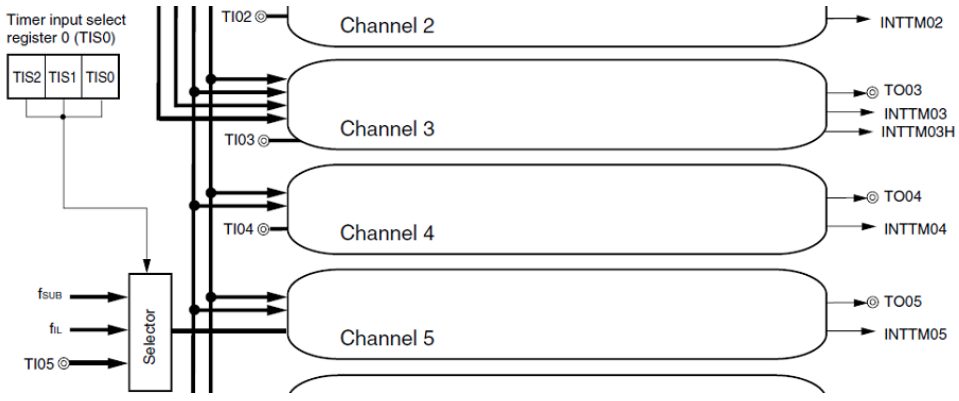

Figure 7 Timer Array Unit Channel 5 Configuration

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

System Clock / (Reference Clock / 2) x 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) x Loop Count

The cycle count can be calculated as shown in the code extract in figure 8 below

½ the reference clock = 15.26uS (32KHz / 2)

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

At 32MHz this is 187.5nS (6 x 31.25nS)

Therefore the approximate software count for the test harness example is 15.26uS / 187nS = 82 (h’52)

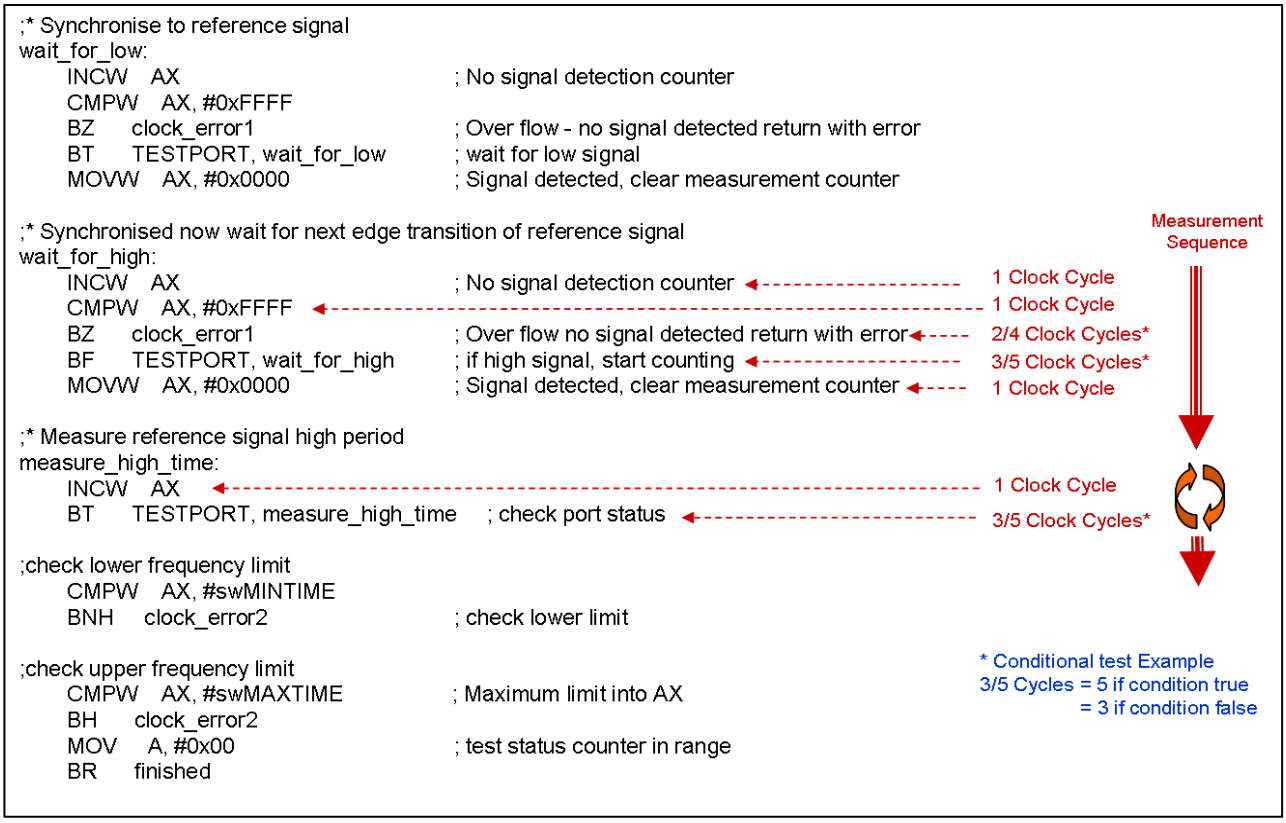

Figure 8 Timer Array Unit Channel 5 Configuration

Table 12: Source files: Software Clock test

STL File name	Header Files
stl_RL78_sw_clocktest.asm	stl_clocktest.h stl.h
Test Harness File Names	**Header Files**
main.c stl_global_data_example.c stl_main_example_support function.c stl_peripheralinit.c	main.h stl_gobal_data_example.h

Declaration	
bool stl_RL78_sw_clocktest(void)	
Description	
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 (stl_clocktest.h), and the result status (Pass / Fail / No reference clock) is returned to the calling function. The reference limits calculation is based on the following System Clock / (Reference Clock / 2) x times the number of clock cycles executed in the count loop The function "indicate_test_result.c" will be called by the test harness control files (main.c) to process the test result Note Function "indicate_test_result.c "is located in the module stl_main_example_support function.c	
Input Parameters	
swMAXTIME	Upper time limit compare value (Defined in stl_clocktest.h)
swMINTIME	Lower time limit compare value (Defined in stl_clocktest.h)
TESTPORT	Test Port Input Pin for external reference signal input (Defined in stl_clocktest.h)
Output Parameters	
NONE	N/A
Return Values	
bool	Test status result returned in CPU register A 0 = Test passed. 1 = Test measurement failed (Outside the reference window) 2 = Test failed (No reference clock detected)

Table 13: Source files: Hardware Clock test

STL File name	Header Files
stl_RL78_hw_clocktest.asm	stl_clocktest.h stl.h
Test Harness File Names	**Header Files**
main.c stl_global_data_example.c stl_main_example_support function.c stl_peripheralinit.c	main.h stl_gobal_data_example.h

Declaration	
bool stl_RL78_hw_clocktest(void)	
Description	
This function tests the system clock using the hardware measurement (TAU channel n) feature. The measured result (capture value) is compared against the upper and lower limit values defined in the clock test header file (stl_clocktest.h) and the result status (Pass / Fail / No reference clock) is returned to the calling function. The function "indicate_test_result.c" will be called by the test harness control files (main.c) to process the test result Note Function "indicate_test_result.c "is located in the module stl_main_example_support function.c	
Input Parameters	
hwMAXTIME	Upper time limit compare value (Defined in stl_clocktest.h)
hwMINTIME	Lower time limit compare value (Defined in stl_clocktest.h)
CAPTURE_interrupt_FLAG	Timer channel Capture Interrupt Flag (Defined in stl_clocktest.h)
CAPTURE_Register_Addr	Timer channel capture register address (Defined in stl_clocktest.h)
Output Parameters	
NONE	N/A
Return Values	
bool	Test status result returned in CPU register A 0 = Test passed. 1 = Test measurement failed (Outside the reference window) 2 = Test failed (No reference clock detected)

3 Example Usage

In addition to the actual test software source files, the IAR Embedded Workbench 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 typically 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. Tests that should be run are

   1. All Ram using Initial March C (or initial March X)
   2. All register tests
   3. Flash Memory CRC Test
   4. The clock test may be run at a later time depending on the initial clock speed if 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.

   1. 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.
   2. Register Tests. These are dependant upon the application timing
   3. 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
   4. 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 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 the complete 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 IAR embedded workbench tool chain can be used to calculate and add a CRC value and placed at a location at a location specified by the user. This can be done via an external utility or using the IAR tool to generate the reference CRC value and place the reference value in memory. Please note that this utility can only be used when using the "software" CRC module. To provide a reference value when using the on chip CRC peripheral, an external CRC program can be used or alternatively an example of an internal application based reference generation is included in the test harness project.

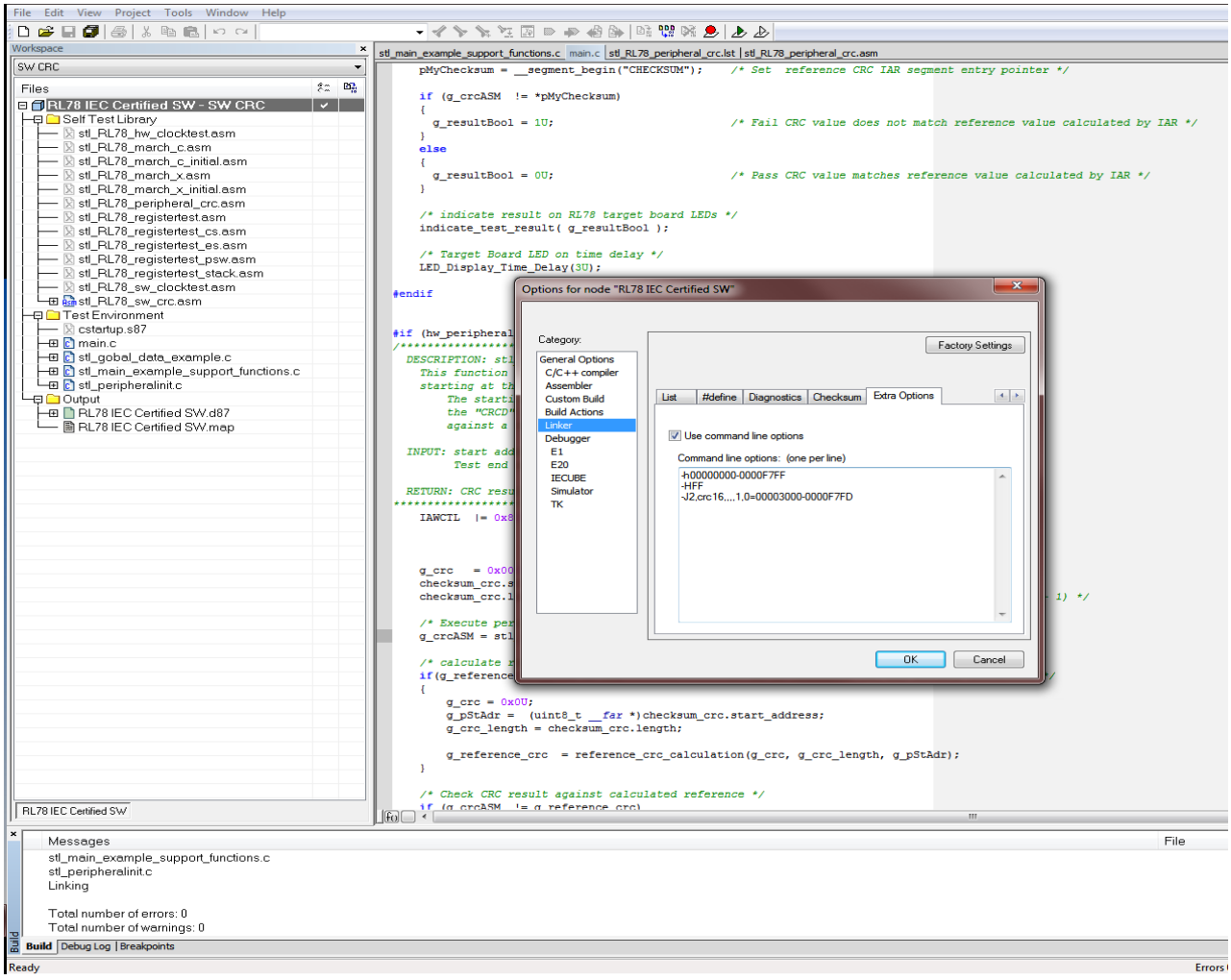

Figure 9: Adding Reference CRC

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. The stack area cannot be tested after the system has been initialised, unless the content is relocated to a new area and the stack pointer changed accordingly. Interrupts should not be serviced 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.

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 the aim of 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 that the system clock in order to increase the accuracy of the clock measurement.

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

4 Benchmarking

4.1 Development Environment

- IECUBE - QB-RL78G13-ZZZ-EE Full RL78/G13 In circuit Emulator
- QB-R5F100LE-TB RL78/G13 Target Board 64pin LQFP (10 x 10mm)
- Tool chain: IAR Embedded Workbench Version 1.20.1

MCU: R5F100LE (64KB Flash, 4KB RAM 64pin)

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

External Sub Clock: 32 KHz

4.2 IAR Embedded Workbench Settings

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

4.2.1 General Options

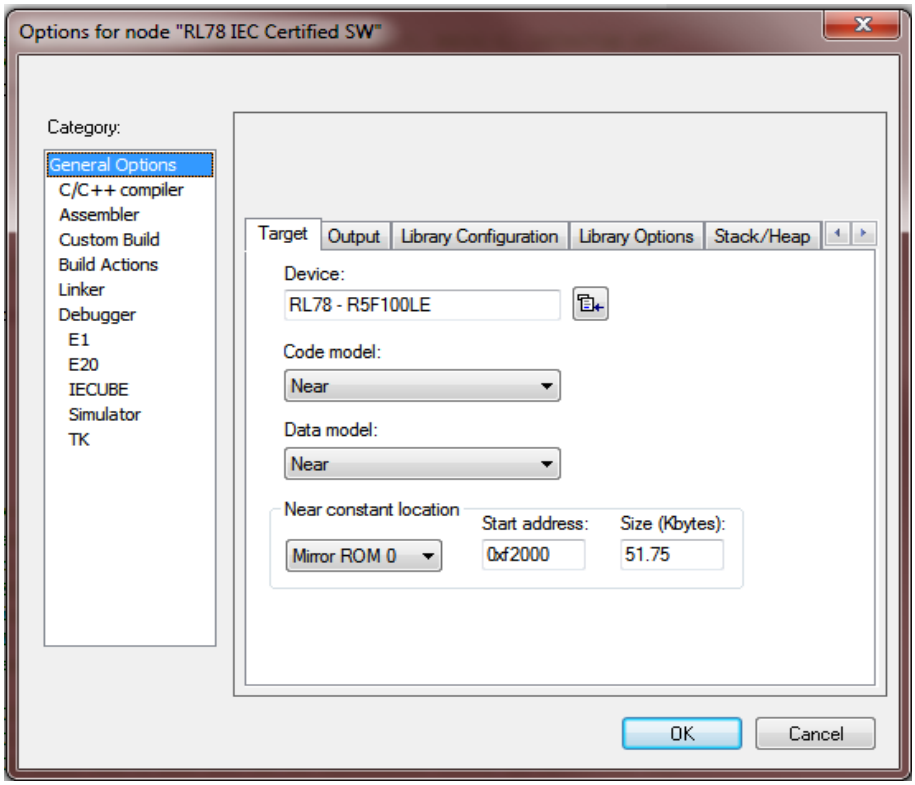

Figure 10 IAR EW General Options - Target Device

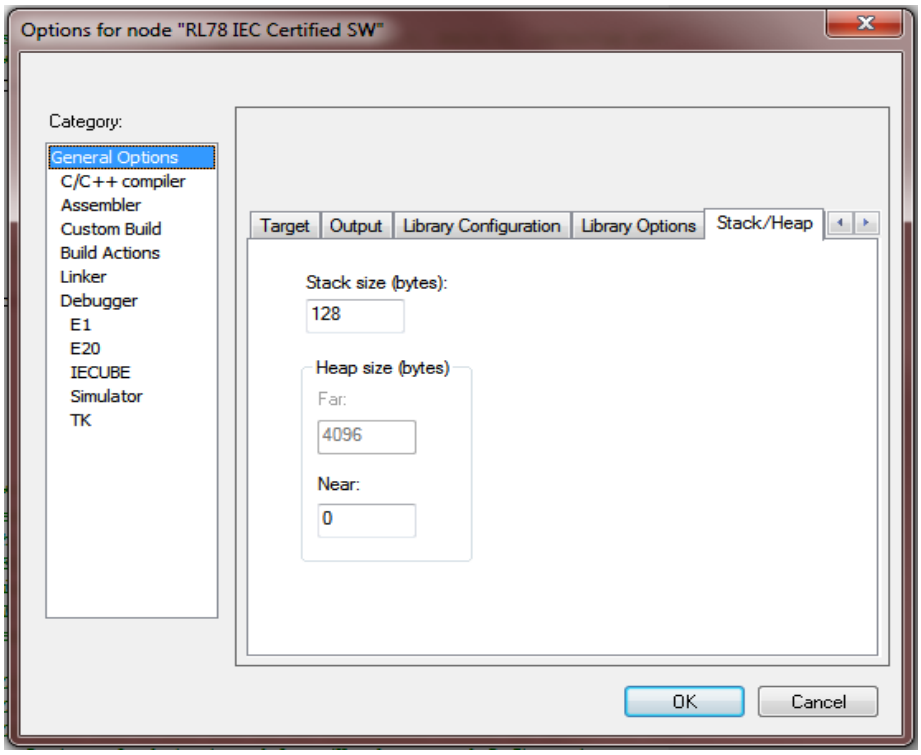

Figure 11 IAR EW General Options – Stack/Heap

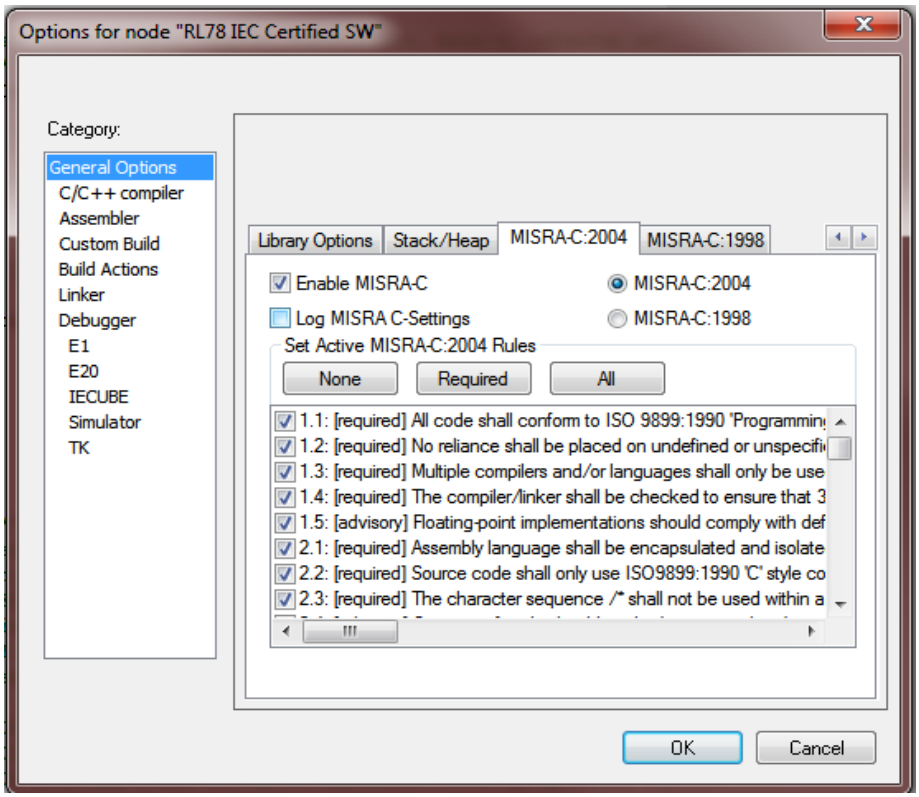

Figure 12 IAR EW General Options – MISRA-C

4.2.2 Complier Settings

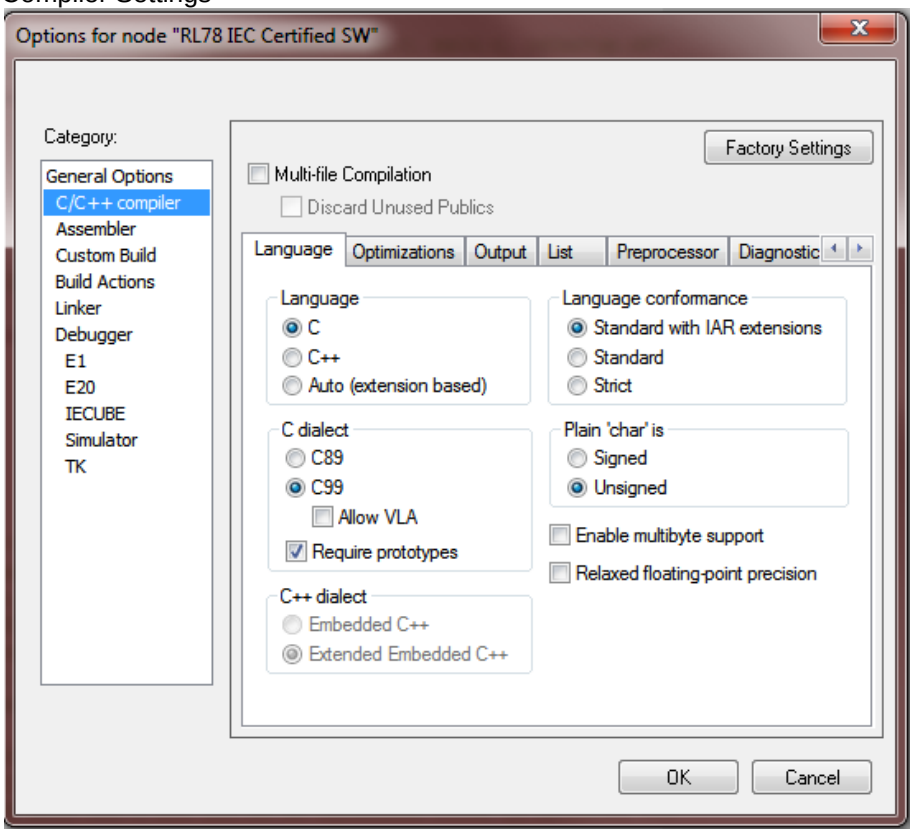

Figure 13 IAR EW Compiler Options - Language

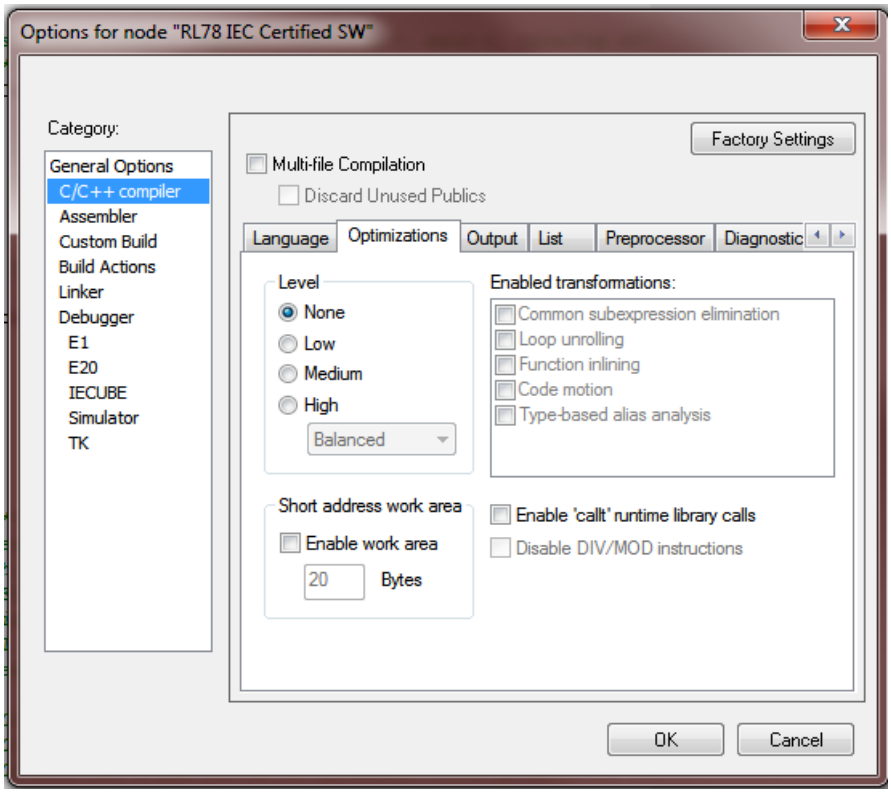

Figure 14 IAR EW Compiler Options - Optimisation

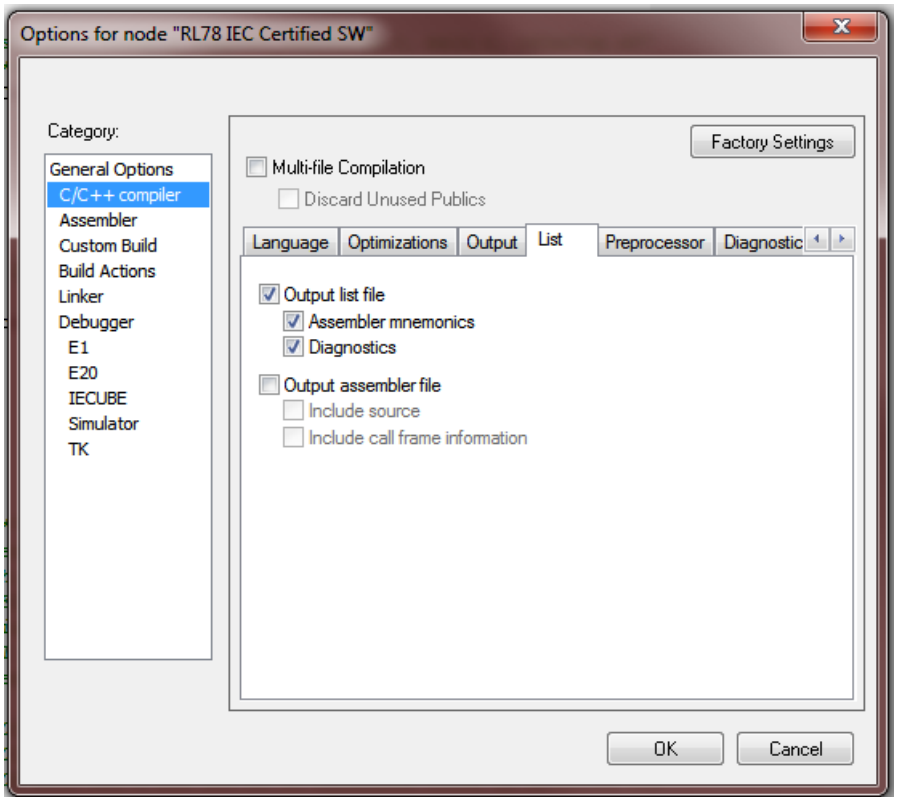

Figure 15 IAR EW Compiler Options - Listings

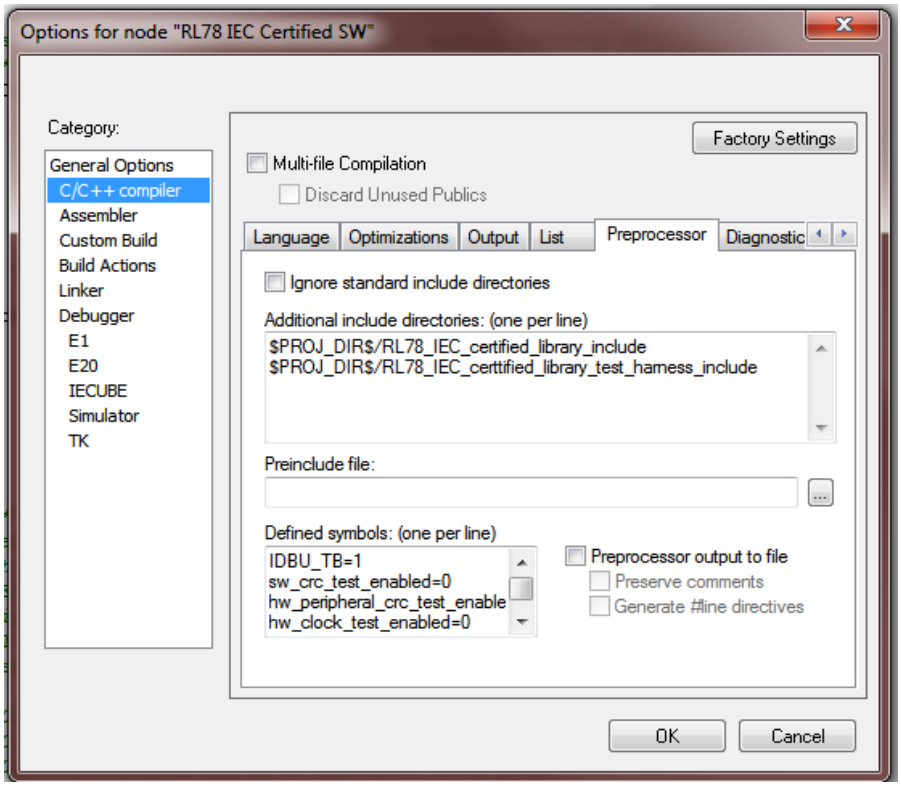

Figure 16 IAR EW Compiler Options – Pre-processor Defined symbols

4.2.2.1 Defined Symbols

The following defined symbols are included in the Compiler options section. Each symbol is preset to enable the conditional build options for each of the self test projects is selected.

Example shown below is for the "System" March C test

Table 14: Compiler Defined Symbols

IDBU_TB=1	Use the RL78/G13 target board as target hardware
sw_crc_test_enabled=0	Set configuration for Software CRC Self Test (disabled)
hw_peripheral_crc_test_enabled=0	Set configuration for Hardware CRC Self Test (disabled)
hw_clock_test_enabled=0	Set configuration for Hardware Clock Self Test (disabled)
sw_clock_test_enabled=0	Set configuration for Software Clock Self Test (disabled)
march_c_test_enabled=1	Set configuration for "System" March C Self Test (Enabled)
march_x_test_enabled=0	Set configuration for "System" March X Self Test (disabled)
register_test_enabled=0	Set configuration for Working Registers Self Test (disabled)
register_test_psw_enabled=0	Set configuration for PSW Register Self Test (disabled)
register_test_stack_enabled=0	Set configuration for Stack Pointer Register Self Test (disabled)
register_test_cs_enabled=0	Set configuration for Extended Code Address Register Self Test (disabled)
register_test_es_enabled=0	Set configuration for Extended Data Address Register Self Test (disabled)

4.2.3 Assembler Settings

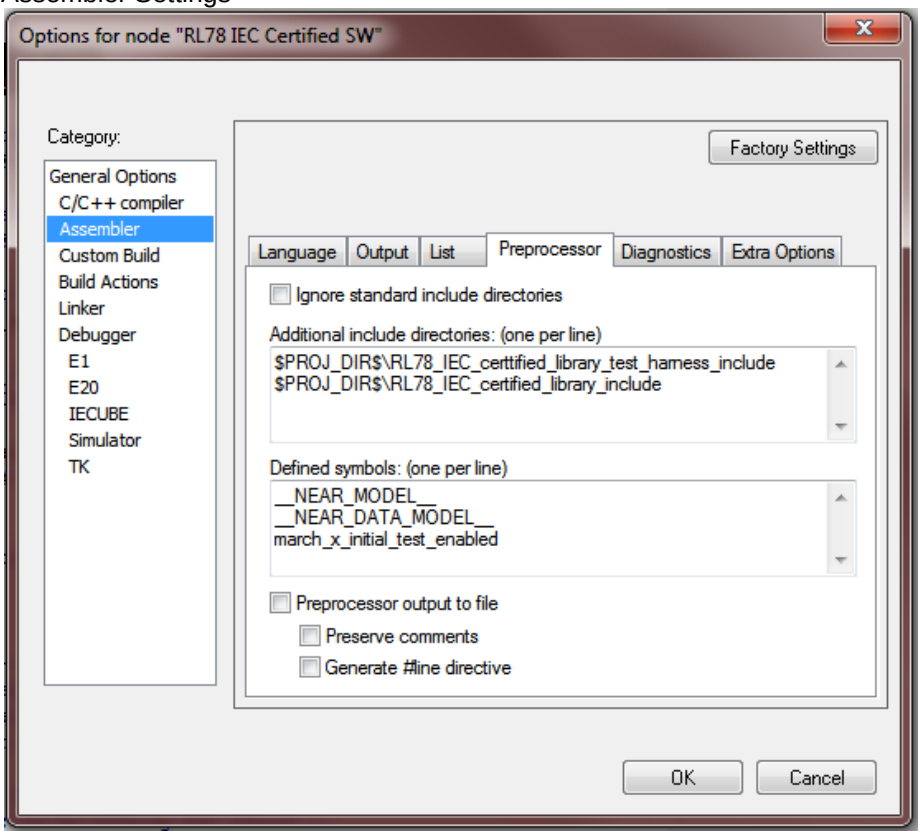

Figure 17 IAR EW Assembler Options – Pre-processor Defined symbols

4.2.3.1 Defined Symbols

The following defined symbols are included in the IAR EW assembler options section.

Each symbol is preset to enable the conditional build options for each of the self test projects selected.

This differs from the compiler settings as if the symbol is omitted then the self test is disabled. If the symbol is included then the test is enabled. The example shown in figure 17 above enables the “initial” March X test

Table 15: Assembler Defined Symbols

March_c_initial_test_enabled	Set configuration for Initial March C test
March_x_initial_test_enabled	Set configuration for Initial March X test

4.2.4 Linker Settings

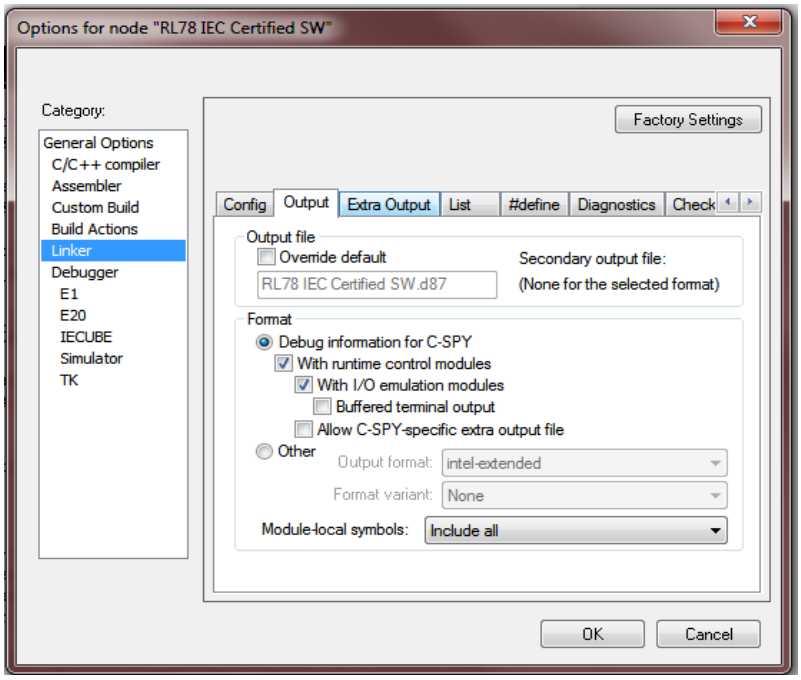

Figure 18 IAR EW Linker Options – Output File settings

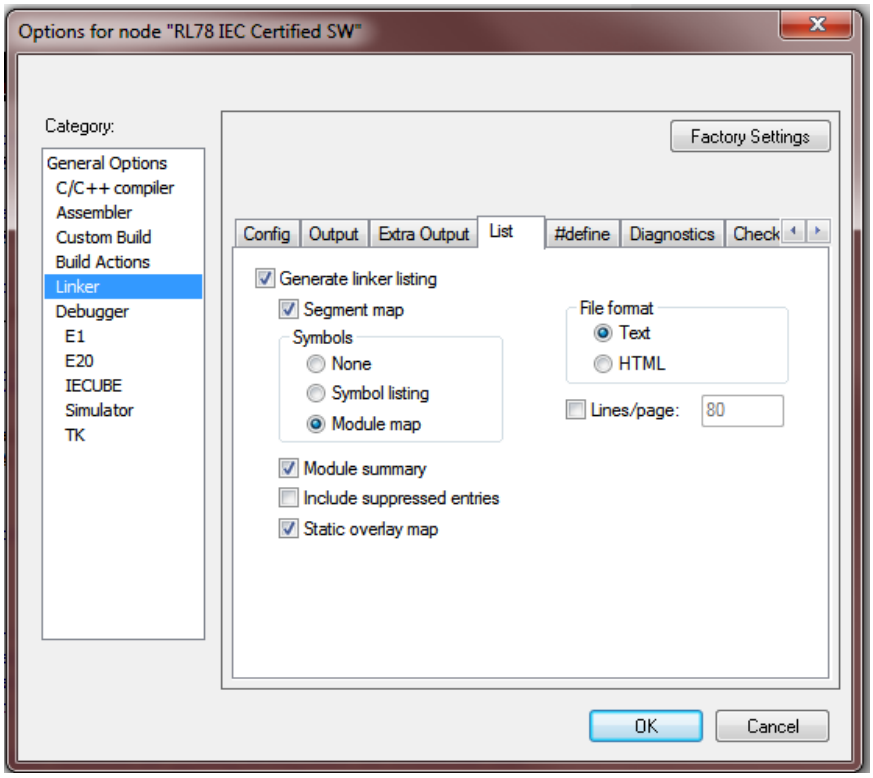

Figure 19 IAR EW linker Options – Listings (Map File)

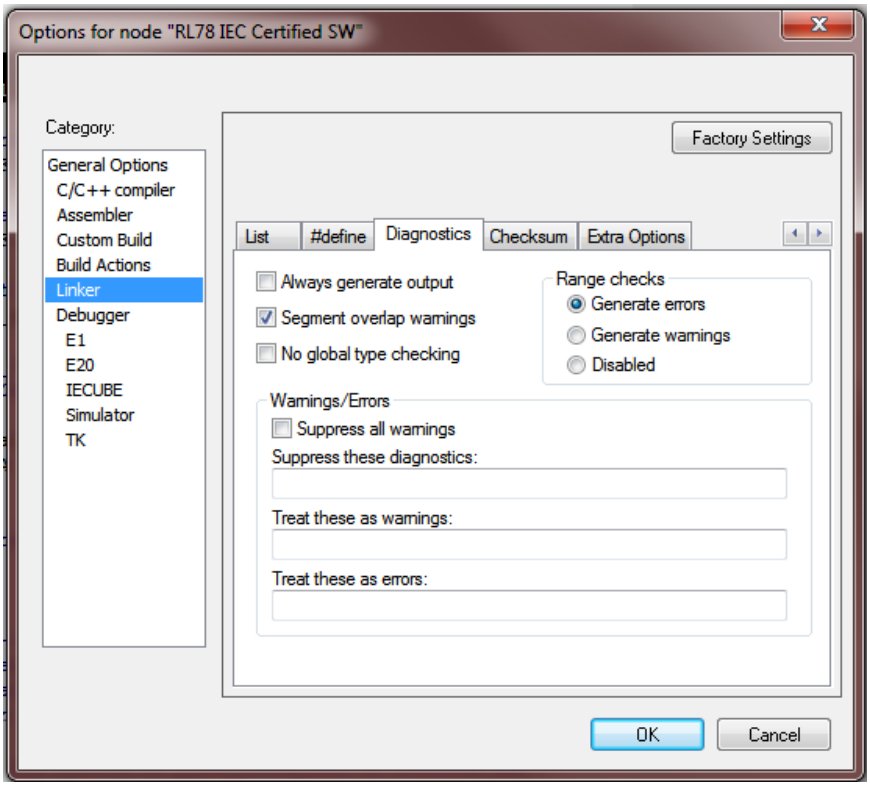

Figure 20 IAR EW linker Options – Diagnostics

5 Resources

The following resource tables include only the resources and execution times needed for the self test modules. The figures do not include data for the test harness files.

All timings are based on the following environment settings

MCU: R5F100LE (64KB Flash, 4KB RAM, 64 pin)

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

External Sub Clock: 32 KHz

3.1 CPU Register Tests

Note: Optimisation should not be used for these tests.

3.1.1 Working Registers Test

Table 16: Working registers Resources

Measurement	
Code size	178B
Stack usage (Module use plus function call)	12B
Data	1B
Execution time	320 cycles
	10uS

3.1.2 PSW Register Test

Table 17: PSW Registers Resources

Measurement	
Code size	43B
Stack usage (Module use plus function call)	6B
Data	1B
Execution time	32 cycles
	1uS

5.1.3 Stack Pointer Register Test

Note: This test cannot be run on the E1 on Chip debugger.

Table 18: Stack Pointer Register Resources

Measurement	
Code size (Bytes)	47
Stack usage (Module use plus function call)	6
Data	1
Execution time	30 cycles
	0.8uS

5.1.4 Code Extension (CS) Register Test

Table 19: Code extension Register Resources

Measurement	
Code size	43B
Stack usage (Module use plus function call)	6B
Data	1B
Execution time	31 cycles
	0.97uS

5.1.5 Data Extension (ES) Register Test

Table 20: Data Extension Register Resources

Measurement	
Code size	41B
Stack usage (Module use plus function call)	6B
Data	1B
Execution time	29 cycles
	0.9uS

5.2 Flash ROM Test

5.2.1 Software CRC test

Table 21: Software CRC Test Resources

Measurement	
Code size plus constants	616B
Stack usage (Module use plus function call)	6B
Data	2B
Execution time (Range = 51KB)	2496000 cycles
	78mS

5.2.2 Hardware CRC test

Table 22: Hardware CRC Test Resources

Measurement	
Code size	77B
Stack usage (Module use plus function call)	6B
Data	0B
Execution time (Range = 61KB)	1408000 cycles
	44mS

5.3 RAM

5.3.1 System March C

Table 23: System March C Test Resources

Measurement	
Code size	354B
Stack usage (Module use plus function call)	18B
Data	0B
Execution time (Range = 1.3KB)	476800 cycles
	14.9mS

5.3.2 System March X

Table 24: System March X Test Resources

Measurement	
Code size	298B
Stack usage (Module use plus function call)	18B
Data	0B
Execution time (Range = 57KB)	243200 cycles
	7.6mS

5.3.3 Initial March C

Table 25: Initial March C Test Resources

Measurement	
Code size	337B
Stack usage (Module use plus function call)	0B
Data	0B
Execution time (Range = 4KB)	1520000 cycles
	47.5mS

5.3.4 Initial March X

Table 26: Initial March X Test Resources

Measurement	
Code size	279B
Stack Usage (Module use plus function call)	0B
Data Usage	0B
Execution time (Range = 4KB)	768000 cycles
	24mS

5.4 Clock Monitor

The execution time of this function is defined by the frequency of the reference clock therefore the execution time is not measured.

Table 27: Software Clock Monitor Test Resources

Measurement	
Code Size: Program + Data	65B
Data Usage	0B
Stack Usage (Module + function call)	8B
Execution time	N/A

Table 28: Hardware Clock Monitor Test Resources

Measurement	
Code Size	62B
Data Usage	0B
Stack Usage (Module + function call)	8B
Execution time	N/A

6 Additional Hardware Resources

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

6.1 Additional Safety Functions

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

6.1.1 RAM 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.

Figure 22-6. 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 21 RAM Parity Error Checking

6.1.2 RAM Guard Protection

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

The Ram area available for this feature is limited and can be selected by the "GRAM0, GRAM1" bits as shown in figure 22 below:

Figure 22-7. 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[Note 1]	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[Note 2]
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 22 RAM Guard Protection

6.1.3 Invalid Memory Access Protection

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

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

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

Figure 22-7. 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[Note 1]	Control of invalid memory access detection
0	Disable the detection of invalid memory access.
1	Enable the detection of invalid memory access.

Figure 23 Invalid Memory Access Protection

6.1.4 I/O Port SFR Protection

This is a write protection feature that prohibits a write to the SFR registers. No error is generated if a write occurs, but the write operation does not change the state of the registers involved.

Please note that the data port register (Pxx) cannot be protected.

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

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

PMxx, PUxx, PIMxx, POMxx, 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 24 below

Figure 22-7. 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[Note 1]	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[Note 2]
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[Note 3]
0	Disabled. Port registers can be read or written to.
1	Enabled. Writing to port registers is disabled. Reading is enabled.

Figure 24 I/O Port SFR Guard Protection

6.1.5 Interrupt SFR Protection

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

The following interrupt registers can be protected with this function

IFxx, MKxx, PRxx, EGPx, and EGNx

The interrupt SFR registers can be guarded by the "GINT" bit as shown in figure 25 below

Figure 22-7. 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[Note 1]	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[Note 2]
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[Note 3]
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[Note 4]
0	Disabled. Interrupt registers can be read or written to.
1	Enabled. Writing to interrupt registers is disabled. Reading is enabled.

Figure 25 Interrupt SFR Guard Protection

6.1.6 Control Register Protection

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

The following control registers can be protected with this function

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

The interrupt SFR registers can be guarded by the "GCSC" bit as shown in figure 26 below

Figure 22-7. 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[Note 1]	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[Note 2]
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[Note 3]
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[Note 4]
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[Notes 5, 6]
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 26 Invalid Memory Access Protection

6.2 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	ANIxx (This is specified using the analog input channel specification register (ADS).)
0	1	0	AV_{REFM}
0	1	1	AV_{REFP}
Other than the above			Setting prohibited

Figure 27 ADC Self Test

7 VDE Certification Status

Module / Version	V1.0	V1.1	V2.0	V3.0
stl_RL78_registertest.asm	Y	--	--	--
stl_RL78_registertest_psw.asm	Y	--	--	--
stl_RL78_registertest_stack.asm	Y	--	--	--
stl_RL78_registertest_cs.asm	Y	--	--	--
stl_RL78_registertest_es.asm	Y	--	--	--
stl_RL78_sw_crc.asm	Y	--	N	--
stl_RL78_peripheral_crc.asm	Y	--	N	--
stl_RL78_march_c.asm	Y	N	N	--
stl_RL78_march_x.asm	Y	--	N	--
stl_RL78_march_c_initial.asm	Y	--	N	--
stl_RL78_march_x_initial.asm	Y	--	N	--
stl_RL78_sw_clocktest.asm	Y	--	--	--
stl_RL78_hw_clocktest.asm	Y	--	N	*

Y = VDE Certified Module

-- = No Code Change affecting VDE Certified Module in this update

N = Module not VDE Certified. Module code changed in this update

* = Module Code Change in this update

Website and Support

Renesas Electronics Website
http://www.renesas.com/

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

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

Revision History

Rev.	Date	Description	
		Section	**Summary**
0.01	Jul 01, 2011	All	Preliminary revision
0.02	Jul 21, 2011	All	Updated and corrected preliminary for review
1.00	July 06, 2011	All	First Release
1.01	Nov 16, 2011	P21 and P42	Text correction - March C corrected to March X.
		P28	Removed error statement. Unknown origin.
		All	Format Update for Headers and footers.
1.02	Feb 01, 2012	P15	Removed text, as no return value required in this table.
		P18 and P19	Pointer changed from __far to normal.
		P22	Correction to system clock test description, Hardware measurement.
		P26	Added “CAPTURE_Register_Addr” definition to Input Parameter section in table.
		P49	Added module certification status
	Jul 15, 2012	P50	Updated Contact details
2.00	May 23, 2013	P31	Update for IAR Embedded Workbench to V1.20.1
		P49	Updated module certification status table
2.01	Mar 04, 2014	P4	Update of IEC Annex H references
		P50	Updated web information
			Updated General Precautions page

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU 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. 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 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. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment 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 moment 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 moment 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 moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has 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. Moreover, 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.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the 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 of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of 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. 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 should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended 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; and industrial robots etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.
   Renesas Electronics products are neither intended nor 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 damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, 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, please evaluate 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. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may 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 should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be 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, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

RENESAS

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 LanGao Rd., Putuo District, Shanghai, China
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

© 2014 Renesas Electronics Corporation. All rights reserved.

Colophon 3.0