To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

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

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. 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.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. 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.
5. When exporting the 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. You should not use Renesas Electronics products or the 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. 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.
6. 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.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. 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 categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. 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 an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   "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.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. 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.
9. 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 system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. 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.
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.

R8C Tiny, M16C Tiny, SH Tiny, and H8 Tiny Series

Self Test Sample Code for Renesas Microcontrollers

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 IEC 60730 safety standard for household appliances requires manufactures to design automatic electronic controls that ensure safe and reliable operation of their products.

The IEC 60730 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 IEC 60730 class B safety standards. Although these routines were developed using IEC 60730 as the basis, they can be implemented in any system for self testing of Renesas MCUs.

These three key components are:

1. CPU
2. ROM / Flash memory
3. RAM

The software routines provided for CPU, ROM and RAM testing can be used after reset and also during the program execution. The end user has the flexibility of how to integrate these routines into their overall system design.

TABLE OF CONTENTS

Introduction 1
1 Tests 4
1.1 CPU test 4
1.1.1 R8C Tiny CPU Tests 5
1.1.2 M16C Tiny CPU Tests 9
1.1.3 SH Tiny CPU Tests 10
1.1.4 H8 Tiny CPU Tests 14
1.2 ROM / Flash memory test 17
1.2.1 CRC16-CCITT 17
1.2.1.1 CRC16-CCITT calculation methods 17
1.2.1.2 Alternative CRC implementations 18
1.2.2 CRC16-CCITT software interface 19
1.3 RAM test 20
1.3.1 Algorithms implemented 20
1.3.1.1 March C 20
1.3.1.2 March X 21
1.3.1.3 March X (Word-Oriented Memory version) 21
1.3.2 Software API 22
1.3.2.1 March C API 22
1.3.2.2 March X API 23
1.3.2.3 March X WOM API 24
2 Development environments 25
2.1 R8C Tiny development environment 26
2.1.1 Tool chain settings 26
2.1.1.1 No optimisation 26
2.1.1.2 Minimal size 26
2.1.1.3 Best speed 26
2.2 SH Tiny development environment 27
2.2.1 Tool chain optimisation settings for Phase 1 27
2.2.1.1 No optimisation 27
2.2.1.2 Minimal size 27
2.2.1.3 Best speed 27
2.2.2 Tool chain optimisation settings for Phase 2 28
2.2.2.1 No optimisation 28
2.2.2.2 Minimal size 28
2.2.2.3 Best speed 28
2.3 M16C Tiny development environment 29
2.3.1 Tool chain optimisation settings 29
2.3.1.1 No optimisation 29
2.3.1.2 Minimal size 29
2.3.1.3 Best speed 29
2.4 H8 Tiny development environment 30
2.4.1 Tool chain optimisation settings for Phase 1 30
2.4.1.1 No optimisation 30
2.4.1.2 Minimal size 30
2.4.1.3 Best speed 30
2.4.2 Tool chain optimisation settings for Phase 2 31
2.4.2.1 No optimisation 31
2.4.2.2 Minimal size 31
2.4.2.3 Best speed 31
3 Benchmarking results 32
3.1 R8C Tiny 33
3.1.1 CPU test results 33
3.1.2 RAM test results 33
3.1.2.1 March C 33
3.1.2.2 March X 34
3.1.2.3 March X WOM 35
3.1.3 ROM test results 36
3.2 SH Tiny 38
3.2.1 CPU test results 38
3.2.2 RAM test results 38

3.2.2.1 March C 38
3.2.2.2 March X 39
3.2.2.3 March X WOM 40
3.2.3 ROM test results 41
3.3 M16C Tiny 43
3.3.1 CPU test results 43
3.3.2 RAM test results 43
3.3.2.1 March C 43
3.3.2.2 March X 44
3.3.2.3 March X WOM 45
3.3.3 ROM test results 46
3.4 H8/Tiny 48
3.4.1 CPU test results 48
3.4.2 RAM test results 48
3.4.2.1 March C 48
3.4.2.2 March X 49
3.4.2.3 March X WOM 50
3.4.3 ROM test results 51
3.5 Results Comparison Tables 51
4 Abbreviations 51
5 Website and Support 51
6 Cautions 51

1 TESTS

1.1 CPU TEST

This section describes CPU tests routines. Reference IEC 60730: 1999+A1:2003 Annex H - Table H.11.12.1 CPU

The CPU test covers testing of General Purpose Registers (GPR), Flag / Status register, Program Counter (PC) register and other platform specific registers.

The test itself will write test values (like 0x55, 0xAA) into the target register and read it back. It is almost impossible to access hardware registers of the MCU using 'C' language, so it has been decided to write inline assembly code for individual platform. Hence the source code will be obviously different from one MCU to another one.

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

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

This will be called by the CPU test if an error is detected. It is thought that this function will not return back to the test code.

The test functions all follow the rules of register preservation following a C function call as specified in the Renesas tool chain manual. Therefore the user can call these functions like any normal C function without any additional responsibilities for saving register values beforehand.

1.1.1 R8C Tiny CPU Tests

The R8C Tiny CPU testing needs to test R0-R3, A0-A1, FB, INTBL, INTBH, USP, ISP, SB and FLG registers. The source file 'CPU_Test.c' provides implementation of CPU test using "C" language with inline assembly for R8C Tiny. The source file 'CPU_Test.h' provides the interface to the function CPU test. The file 'MisraTypes.h' includes definitions of MISRA compliant standard data types.

IMPORTANT NOTE: Please keep the "Optimisation" option "OFF" for the 'CPU_Test.c' file.

The CPU test is categorised as follows:

- General purpose registers (R0,R1, R2, R3,A0, A1, FB).
- Interrupt Table registers (INTBL, INTBH).
- User Stack Pointer (USP), Interrupt Stack Pointer (ISP), Static Base (SB) register.
- Program counter (PC) register.
- Flag (FLG) or Status register.
- Test all above

Syntax	
`void CPU_TestAll(void)`	
Description	
Runs through all the tests detailed below in the following order:- 1. TestGPRs 2. TestIntRegs 3. TestStackRegs 4. TestPCReg 5. TestFlagReg It is the calling function's responsibility to ensure no interrupts occur during this test. If an error is detected then external function 'ErrorHandler' will be called. See the individual tests for a full description.	
Input Parameters	
`NONE`	N/A
Output Parameters	
`NONE`	N/A
Return Values	
`NONE`	N/A

Syntax	
`void TestGPRs(void)`	
Description	
This function provides CPU General Purpose Registers Testing to test R0, R1, R2, R3, A0, A1 & FB in Register Bank0 and Bank1.	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

Syntax	
`void TestIntRegs(void)`	
Description	
This function provides Interrupt Table (INTBL, INTBH) register testing. **Assumption**: R0 is used as a utility register in this test. This test will fail if R0 is faulty.	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

Syntax
`void TestStackRegs(void)`
Description
This function provides Stack (USP, ISP) and Static Base (SB) Registers testing. **Assumption**: R0 and R1 are used as utility registers in this test. This test will fail if R0 or R1 are faulty.
Input Parameters
`NONE` — N/A
Output Parameters
`NONE` — N/A
Return Values
`NONE` — N/A

Syntax
`void TestFlagReg(void)`
Description
This function provides the flag (FLG) register testing. **Assumption**: R0 is used as a utility register in this test. This test will fail, if R0 is faulty.
Input Parameters
`NONE` — N/A
Output Parameters
`NONE` — N/A
Return Values
`NONE` — N/A

Syntax
`bool_t TestPCReg(void)`
Description
This function provides the Program Counter (PC) register test. This provides a confidence check that the PC is working. It tests that the PC is working by calling a function that is located in its own section so that it can be located away from this function, so that when it is called more of the PC Register bits are required for it to work. So that this function can be sure that the function has actually been executed it returns the inverse of the supplied parameter. This can be checked for correctness only. If an error is detected then external function 'ErrorHandler' will be called.

Input Parameters	
`NONE`	N/A
Output Parameters	
`NONE`	N/A
Return Values	
`bool_t`	TRUE - Success, FALSE - Fail

1.1.2 M16C Tiny CPU Tests

The M16C Tiny and R8C Tiny devices have an identical set of CPU registers.
Please refer to §1.1.1 on page 5 for details of the CPU tests.

1.1.3 SH Tiny CPU Tests

The SH Tiny CPU Tests consist of the files cpu_test_sh7124.c and cpu_test_sh7124.h. Despite it being a 'C' file and the function APIs being 'C' the majority of the code is written in assembly.

The `'#pragma inline_asm'` directive is used to embed the assembly in the C.

The API allows a number of tests to be selected individually but it also provides a function called 'CPU_Test_ALL' which will go through all of the individual tests in sequence. It is recommended that interrupts are disabled before calling any of these tests. However it is only test 'CPU_Test_GBR_VBR_SR' and consequentially 'CPU_Test_All', where it is imperative that no interrupts occur.

The read/write bits of the following 32 bit registers are tested using a simple write then read back test using values h'AAAAAAAA and then h'55555555.

- General Purpose Registers R0 to R15
- System Registers MACH, MACL and PR.
- Control Registers SR, GBR and VBR.

See individual test descriptions below for details:

Syntax	
`void CPU_Test_ALL(void)`	
Description	
Runs through all the tests detailed below in the following order:- 1. CPU_Test_R0toR7 2. CPU_Test_R8toR15 3. CPU_Test_MACH_MACL_PR 4. CPU_Test_GBR_VBR_SR 5. CPU_Test_PC It is the calling functions responsibility to ensure no interrupts occur during this test. If an error is detected then external function CPU_Test_ErrorHandler will be called. See the individual tests for a full description.	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

Syntax
`void CPU_Test_R0toR7(void)`
Description
Test registers R0 to R7. Registers are tested in pairs, for each pair of registers:- 1. Write h'55555555 to both. 2. Read both and check they are equal. 3. Write h'AAAAAAAA to both. 4. Read both and check they are equal. The rules for register preservation state that registers R0 to R7 are free for a function to use without preserving their values. Hence this function does not preserve the values of registers R0 to R7. It is recommended that interrupts are prevented during this test. If an error is detected then external function 'CPU_Test_ErrorHandler' will be called.

Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

Syntax
`void CPU_Test_R8toR15(void)`
Description
Test registers R8 to R15. Registers are tested in pairs, for each pair of registers:- 1. Write h'55555555 to both. 2. Read both and check they are equal. 3. Write h'AAAAAAAA to both. 4. Read both and check they are equal. The rules for register preservation state that registers R8 to R15 must be preserved by a function. Hence this function preserves the values of R8 to R15 It assumes registers RO to R7 are working to achieve this. It is recommended that interrupts are prevented during this test. If an error is detected then external function 'CPU_Test_ErrorHandler' will be called.

Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

Syntax
`void CPU_Test_MACH_MACL_PR (void)`
Description
Test registers MACH, MACL and PR. It assumes registers RO to R7 are working. Register values of MACH, MACL and PR are preserved. For each register:- 1. Write h'55555555 to. 2. Read and check value equalls h'55555555. 3. Write h'AAAAAAAA to. 4. Read and check value equals h'AAAAAAAA. It is recommended that interrupts are prevented during this test. If an error is detected then external function 'CPU_Test_ErrorHandler' will be called.

Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

Syntax
`void CPU_Test_GBR_VBR_SR (void)`
Description
Test registers GBR, VBR and SR. It is the calling functions responsibility to ensure no interrupts occur during this test. It assumes registers RO to R7 are working. Register values of GBR, VBR and SR are preserved. For GBR and VBR registers:- 1. Write h'55555555 to. 2. Read and check value equals h'55555555. 3. Write h'AAAAAAAA to. 4. Read and check value equals h'AAAAAAAA. For SR not all bits are read/write so other test values have been chosen. Also to avoid this function accidentally allowing an interrupt to occur the I bits of the SR are not tested. If an error is detected then external function 'CPU_Test_ErrorHandler' will be called.

Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

Syntax
`void CPU_Test_PC(void)`
Description
Program Counter (PC) register test. This provides a confidence check that the PC is working. It tests that the PC is working by calling a function that is located in its own section so that it can be located away from this function, so that when it is called more of the PC Register bits are required for it to work. So that this function can be sure that the function has actually been executed it returns the inverse of the supplied parameter. This can be checked for correctness only. If an error is detected then external function 'CPU_Test_ErrorHandler' will be called.

Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

1.1.4 H8 Tiny CPU Tests

The H8 Tiny CPU testing needs to test ER0-ER7, PC and CCR (Condition Code Register or Status) registers.

The source file 'CPU_Test.c' provides implementation of CPU test using "C" language with inline assembly for H8 Tiny. The source file 'CPU_Test.h' provides interface to the function CPU test. The file 'MisraTypes.h' includes definitions of MISRA compliant standard data types.

The `'#pragma inline_asm'` directive is used to embed the assembly in the C.

IMPORTANT NOTE: Please keep "Optimisation" option "OFF" for the 'CPU_Test.c' file.

The CPU test is categorised as follows:

- General purpose registers (ER0 to ER7).
- Program counter (PC) register.
- CCR (Condition Code Register or Status) register.
- Test all above

Syntax	
`void CPU_TestAll(void)`	
Description	
Runs through all the tests detailed below in the following order:- 6. CPU_Test_ER0toER7 7. CPU_Test_SR 8. CPU_Test_PC It is the calling function's responsibility to ensure no interrupts occur during this test. If an error is detected then external function CPU_Test_ErrorHandler will be called. See the individual tests for a full description.	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

Syntax
`void CPU_Test_ER0toER7 (void)`
Description
This function provides CPU General Purpose Registers Testing to test ER0 to ER7. For each of register:- 1. Store original contents of the register 2. Write H'55555555 to the register. 3. Read and check the value is are equal. 4. Write H'AAAAAAAA to the register. 5. Read and check the value is are equal. 6. Restore original contents of the register If an error is detected then external function CPU_Test_ErrorHandler will be called.

Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

Syntax
`void CPU_Test_PC (void)`
Description
This function provides the Program Counter (PC) register test. This provides a confidence check that the PC is working. It tests that the PC is working by calling a function that is located in its own section so that it can be located away from this function, so that when it is called more of the PC Register bits are required for it to work. So that this function can be sure that the function has actually been executed it returns the inverse of the supplied parameter. This can be checked for correctness only. If an error is detected then external function 'CPU_Test_ErrorHandler' will be called.

Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

Syntax	
`void CPU_Test_SR (void)`	
Description	
Tests status registers SR / CCR (Condition-Code Register). The original contents of SR are preserved during this test. For CCR register:- 1. Write h'D5 to CCR. (See code for bit detiails) 2. Read back and check value equals h'D5. 3. Write h'AA to CCR. (See code for bit detiails) 4. Read back and check value equals h'AA. If an error is detected then external function CPU_Test_ErrorHandler will be called.	
Input Parameters	
NONE	N/A
Output Parameters	
NONE	N/A
Return Values	
NONE	N/A

1.2 ROM / FLASH MEMORY TEST

This section describes the ROM / Flash memory test using CRC routines. Reference IEC 60730: 1999+A1:2003 Annex H – H2.19.4.1 CRC – Single Word.

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

A CRC checksum is the remainder of a binary division with no bit carry (XOR 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.

During the memory self test the same algorithm generates another checksum, which is compared with the saved checksum. The technique recognises all one-bit errors and a high percentage of multi-bit errors.

CRCs are popular because they are simple to implement in binary hardware and are easy to analyse mathematically.

1.2.1 CRC16-CCITT

The 16-bit CRC16-CCITT specification is:

- Width = 16 bits
- Truncated polynomial = 0x1021
- Initial value = 0xFFFF
- Input data is NOT reflected
- Output CRC is NOT reflected
- No XOR is performed on the output CRC

Advantage of using the 16-bit CRC16-CCITT:

- It is a straightforward 16-bit CRC implementation in that it doesn't involve:
  - reflection of data
  - reflection of the final CRC value
- Starts with a non-zero initial value -- leading zero bits cannot affect the CRC16 used by LHA, ARC, etc., because the initial value is zero.
- It requires no additional XOR operation after everything else is done.

The polynomial for the CRC16-CCITT is 0x1021 ($x^{16} + x^{12} + x^5 + 1$). The initial value for the CRC16-CCITT is 0xFFFF. This value is known as the “seed” value.

1.2.1.1 CRC16-CCITT calculation methods

1.2.1.1.1 Software

The CRC value can be calculated in software using two methods: Look-up table, and bit shifting.

The look-up table method requires more ROM space (512 bytes for 256 16-bit words) than the bit shifting method but uses fewer CPU clock cycles.

1.2.1.1.2 M16C hardware CRC16-CCITT module

The M16C microcontroller series has a CRC calculation circuit that generates the CRC code for one byte of data in two machine clock cycles. In M16C/Tiny the bit order is selectable. The block diagram for M16C/Tiny is shown below.

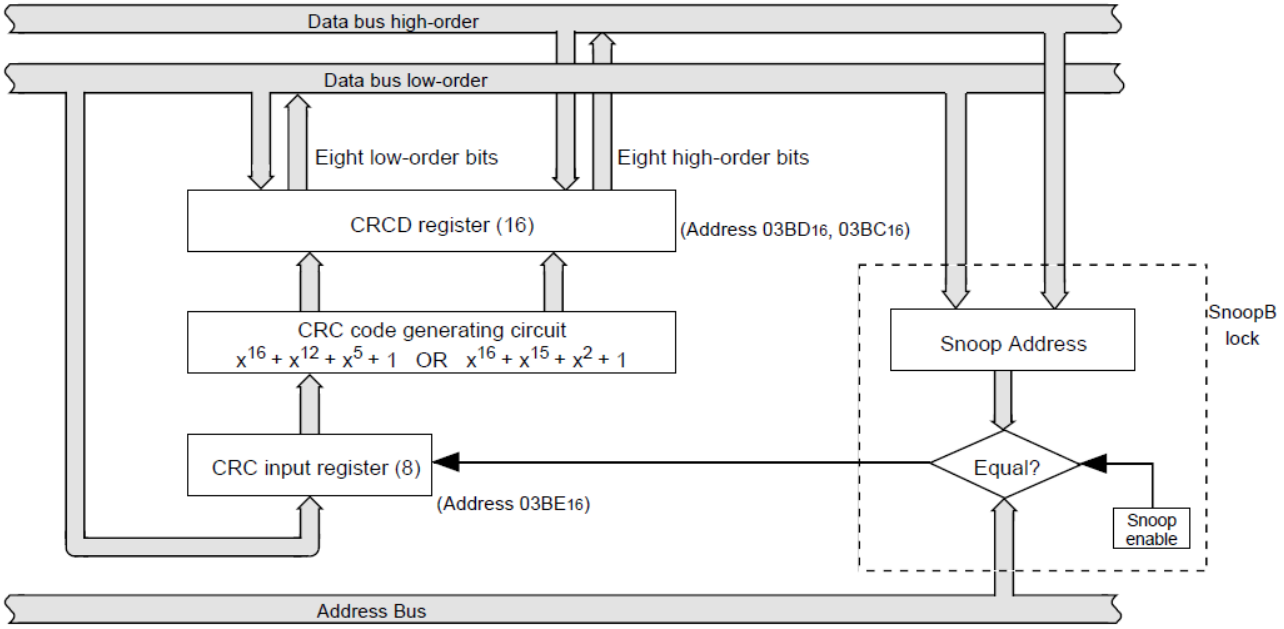


Figure 1: M16C/Tiny CRC16-CCITT circuit

1.2.1.2 Alternative CRC implementations

The look-up table and bit shifting methods described in §1.2.1.1 provide routines optimised for speed or size respectively.
Two more CRC implementations have been developed using smaller look-up tables for applications which require an intermediate solution.

1.2.1.2.1 CRC16-CCITT using a small lookup table

This algorithm requires a smaller static lookup table of sixteen 16-bit words (32 bytes). It computes a checksum for each four bits of the data in a loop.

1.2.1.2.2 CRC16 using a small lookup table

This alternative algorithm, used by encryption routines such as LHA and ARC, also requires a smaller static lookup table requiring only 32 bytes.

The divisor polynomial is now $x^{16} + x^{15} + x^2 + 1$.

1.2.2 CRC16-CCITT software interface

The source file 'CRC16-CCITT.c' provides implementation of CRC16_CCITT using the "C" programming language. The source file 'CRC16-CCITT.h' provides an interface to the function CRC16_CCITT. The file 'MisraTypes.h' includes definitions of MISRA-compliant standard data types.

Syntax	
`uint16_t CRC16_CCITT(const uint8_t* const pui8_DataBuf, const uint32_t ui32_DataBufSize)`	
Description	
This function provides an interface to calculate the CRC16–CCITT using the polynomial ($x^{16} + x^{12} + x^5 + 1$).	
Input Parameters	
`uint8_t* const pui8_DataBuf`	Pointer to start of data buffer / memory. This should be unsigned integer constant pointer to the constant data.
`const uint32_t ui32_DataBufSize`	Length of the data buffer / memory. This should be a 32-bit constant.
Output Parameters	
`NONE`	N/A
Return Values	
`uint16_t`	16-bit calculated CRC16 value.

1.3 RAM TEST

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, while 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) but the supplied test functions provide a non-destructive option so that memory contents can be preserved. This is achieved by copying the memory to a supplied buffer before running the actual algorithm and then restoring the memory from the buffer at the end of the test.

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

1.3.1 Algorithms implemented

1.3.1.1 March C

The March C algorithm (van de Goor 1991) consists of 6 March elements with a total of 11 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 decoder:
   - With a certain address, no cell will be accessed.
   - A certain cell is never accessed.
   - With a certain address, multiple cells are accessed simultaneously.
   - A certain cell can be accessed by multiple addresses.

These are the 6 March elements:-

I. Write all zeros to array
II. Starting at lowest address, read zeros, write ones, increment up array bit by bit.
III. Starting at lowest address, read ones, write zeros, increment up array bit by bit.
IV. Starting at highest address, read zeros, write ones, decrement down array bit by bit.
V. Starting at highest address, read ones, write zeros, decrement down array bit by bit.
VI. Read all zeros from array.

1.3.1.2 March X

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

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

These are the 4 March elements:-

I. Write all zeros to array
II. Starting at lowest address, read zeros, write ones, increment up array bit by bit.
III. Starting at highest address, read ones, write zeros, decrement down array bit by bit.
IV. Read all zeros from array.

1.3.1.3 March X (Word-Oriented Memory version)

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

These are the 6 March elements:-

I. Write all zeros to array
II. Starting at lowest address, read zeros, write ones, increment up array word by word.
III. Starting at highest address, read ones, write zeros, decrement down word by word.
IV. Starting at lowest address, read zeros, write h'AAs, increment up array word by word.
V. Starting at highest address, read h'AAs, write h'55s, decrement down word by word.
VI. Read all h'55s from array.

1.3.2 Software API

1.3.2.1 March C API

The following implementations of the test are available depending upon the required memory access width.

Table 1: Source files:

Access Width	File name
8 Bit	ramtest_march_c_8bit.c and .h
16 Bit	ramtest_march_c_16bit.c and .h
32 Bit	ramtest_march_c_32bit.c and .h

The source is written in ANSI C and uses MISRA-compliant data types as declared in file MisraTypes.h.

Declaration	

`bool_t RamTest_March_C_XXBit(const uint32_t ui32_StartAddr,`
`  const uint32_t ui32_EndAddr,`
`  uintXX_t* const puiXX_RAMSafe);`
 	
`Where XX is the selected memory access width (8, 16 or 32).`	

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

1.3.2.2 March X API

The following implementations of the test are available depending upon the required memory access width.

Table 2: Source files:

Access Width	File name
8 Bit	ramtest_march_x_8bit.c and .h
16 Bit	ramtest_march_x_16bit.c and .h
32 Bit	ramtest_march_x_32bit.c and .h

The source is written in ANSI C and uses MISRA-compliant data types as declared in file MisraTypes.h.

Declaration	

`bool_t RamTest_March_X_XXBit(const uint32_t ui32_StartAddr,`
`const uint32_t ui32_EndAddr,`
`uintXX_t* const puiXX_RAMSafe);`
 	
`Where XX is the selected memory access width (8, 16 or 32).`	

Description	
RAM memory test using March X algorithm.	
Input Parameters	
`ui32_StartAddr`	Address of the first word of RAM to be tested. This must be aligned with the selected memory access width.
`ui32_EndAddr`	Address of the last word of RAM to be tested. This must be aligned with the selected memory access width and be a value greater or equal to ui32_StartAddr.
`puiXX_RAMSafe`	For a destructive memory test set to NULL. For a non-destructive memory test, set to the start of a buffer that is large enough to copy the contents of the test area into it and that is aligned with the selected memory access width.
Output Parameters	
`NONE`	N/A
Return Values	
`bool_t`	TRUE = Test passed. FALSE = Test or parameter check failed.

1.3.2.3 March X WOM API

The following implementations of the test are available depending upon the required memory access width.

Table 3: Source files:

Access Width	File name
8 Bit	ramtest_march_x_wom_8bit.c and .h
16 Bit	ramtest_march_x_wom_16bit.c and .h
32 Bit	ramtest_march_x_wom_32bit.c and .h

The source is written in ANSI C and uses MISRA-compliant data types as declared in file MisraTypes.h.

Declaration	

`bool_t RamTest_March_X_WOM_XXBit(const uint32_t ui32_StartAddr,`
`const uint32_t ui32_EndAddr,`
`uintXX_t* const puiXX_RAMSafe);`
 	
`Where XX is the selected memory access width (8, 16 or 32).`	

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

2 DEVELOPMENT ENVIRONMENTS

The tests were run with optimisation set for none, minimal size or maximum speed.

The reduced-memory CRC algorithms and the March X algorithms were developed in a second phase (referred to as Phase 2). The SH and H8 tool chains had been updated and so there are two sets of settings for these tool chains.

2.1 R8C TINY DEVELOPMENT ENVIRONMENT

Development board: RSK-R8C25, 20 MHz external clock.
MCU: RF21256, configured for 20 MHz internal clock.
Tool chain: Renesas M16C standard tool chain v5.42.00.
In-circuit debugger: E8.

2.1.1 Tool chain settings

2.1.1.1 No optimisation

Compiler: -c -finfo -dir "$(CONFIGDIR)" -Wall -R8C25 -O5OA

Linker: -L "r8clib" -G -MS -O "$(CONFIGDIR)\$(PROJECTNAME).x30" -JOPT -R8C

2.1.1.2 Minimal size

Compiler: -c -finfo -dir "$(CONFIGDIR)" -dSL -O5 -OR -O5OA -OGJ -OSA -fCE -fD32 -fNA -fNC -fSA -fUD -Wall -R8C25

Linker: -L "r8clib" -G -MS -O "$(CONFIGDIR)\$(PROJECTNAME).x30" -JOPT -R8C

2.1.1.3 Best speed

Compiler: -c -finfo -dir "$(CONFIGDIR)" -dSL -O5 -OS -OFFTI -OGJ -OSA -OSTI -OLU=10 -fCE -fD32 -fNC -fSA -fUD -Wall -R8C25 -O5OA

Linker: -L "r8clib" -G -MS -O "$(CONFIGDIR)\$(PROJECTNAME).x30" -JOPT -R8C

2.2 SH TINY DEVELOPMENT ENVIRONMENT

Development board: RSK-SH7124, 10 MHz external clock.
MCU: RF571243, configured for I = P = B = MP = 40MHz.

2.2.1 Tool chain optimisation settings for Phase 1

Tool chain: Renesas SHC standard tool chain **v9.0.3.0**.
In-circuit debugger: E8.

2.2.1.1 No optimisation

Compiler: -cpu=sh2 -define=HMON_Debug -object="$(CONFIGDIR)\$(FILELEAF).obj" -debug -show=tab=4 -optimize=0 -noinline -gbr=auto -chgincpath -errorpath -global_volatile=0 -opt_range=all -infinite_loop=0 -del_vacant_loop=0 -struct_alloc=1 –nologo

Linker: -binary="$(PROJDIR)\7124.cde"(uGenU) -noprelink -rom=D=R -nomessage -list="$(CONFIGDIR)\$(PROJECTNAME).map" -show=symbol -nooptimize -start=DVECTTBL,DINTTBL/00,PIntPRG/0400,CUser_Vectors/0800,uGenU/01000,PHMON,CHMON/03000,PResetPRG,P,C$DSEC,C$BSEC,D,PRAM_TEST_32,PRAM_TEST_16/06000,B,R/0FFFFB000,BHMON/0FFFFBB00,S/0FFFFBE00 -nologo -stack -output="$(CONFIGDIR)\$(PROJECTNAME).abs" -end -input="$(CONFIGDIR)\$(PROJECTNAME).abs" -form=stype -record=s3 -output="$(CONFIGDIR)\$(PROJECTNAME).mot" -exit

2.2.1.2 Minimal size

Compiler: -cpu=sh2 -define=HMON_Debug -object="$(CONFIGDIR)\$(FILELEAF).obj" -debug -show=tab=4 -size -noinline -gbr=auto -chgincpath -errorpath -global_volatile=0 -opt_range=all -infinite_loop=0 -del_vacant_loop=0 -struct_alloc=1 –nologo

Linker: -binary="$(PROJDIR)\7124.cde"(uGenU) -noprelink -rom=D=R -nomessage -list="$(CONFIGDIR)\$(PROJECTNAME).map" -show=symbol -nooptimize -start=DVECTTBL,DINTTBL/00,PIntPRG/0400,CUser_Vectors/0800,uGenU/01000,PHMON,CHMON/03000,PResetPRG,P,C$DSEC,C$BSEC,D,PRAM_TEST_32,PRAM_TEST_16/06000,B,R/0FFFFB000,BHMON/0FFFFBB00,S/0FFFFBE00 -nologo -stack -output="$(CONFIGDIR)\$(PROJECTNAME).abs" -end -input="$(CONFIGDIR)\$(PROJECTNAME).abs" -form=stype -record=s3 -output="$(CONFIGDIR)\$(PROJECTNAME).mot" –exit

2.2.1.3 Best speed

Compiler: -cpu=sh2 -define=HMON_Debug -object="$(CONFIGDIR)\$(FILELEAF).obj" -debug -show=tab=4 -speed -noinline -gbr=auto -chgincpath -errorpath -global_volatile=0 -opt_range=all -infinite_loop=0 -del_vacant_loop=0 -struct_alloc=1 –nologo

Linker: -binary="$(PROJDIR)\7124.cde"(uGenU) -noprelink -rom=D=R -nomessage -list="$(CONFIGDIR)\$(PROJECTNAME).map" -show=symbol -optimize=speed -start=DVECTTBL,DINTTBL/00,PIntPRG/0400,CUser_Vectors/0800,uGenU/01000,PHMON,CHMON/03000,PResetPRG,P,C$DSEC,C$BSEC,D,PRAM_TEST_32,PRAM_TEST_16/06000,B,R/0FFFFB000,BHMON/0FFFFBB00,S/0FFFFBE00 -nologo -stack -output="$(CONFIGDIR)\$(PROJECTNAME).abs" -end -input="$(CONFIGDIR)\$(PROJECTNAME).abs" -form=stype -record=s3 -output="$(CONFIGDIR)\$(PROJECTNAME).mot" -exit

2.2.2 Tool chain optimisation settings for Phase 2

Tool chain: Renesas SHC standard tool chain **v9.1.1.0.**
In-circuit debugger: E10Lite.

2.2.2.1 No optimisation

Compiler: -cpu=sh2
-include="$(WORKSPDIR)\..\..\..\SharedTests\ROM","$(WORKSPDIR)\..\..\..\SharedTests"
-object="$(CONFIGDIR)\$(FILELEAF).obj" -optimize=0 -noinline -gbr=auto -chgincpath -errorpath
-global_volatile=0 -opt_range=all -infinite_loop=0 -del_vacant_loop=0 -struct_alloc=1 -nologo

Linker -noprelink -nodebug -rom=D=R -nomessage -list="$(CONFIGDIR)\$(PROJECTNAME).map"
-show=symbol,reference,xreference -nooptimize
-start=DVECTTBL,DINTTBL/00,PIntPRG/0400,PResetPRG,P,PMY_CRC_CCITT,CMY_CRC_CCITT,P
MY_CRC16,CMY_CRC16,C,C$DSEC,C$BSEC,D/0800,B,R/0FFFFA000,S/0FFFFBE00 -nologo -stack
-output="$(CONFIGDIR)\$(PROJECTNAME).abs" -end
-input="$(CONFIGDIR)\$(PROJECTNAME).abs" -form=stype
-output="$(CONFIGDIR)\$(PROJECTNAME).mot" -exit

2.2.2.2 Minimal size

Compiler: -cpu=sh2
-include="$(WORKSPDIR)\..\..\..\SharedTests\ROM","$(WORKSPDIR)\..\..\..\SharedTests"
-object="$(CONFIGDIR)\$(FILELEAF).obj" -size -noinline -gbr=auto -chgincpath -errorpath
-global_volatile=0 -opt_range=all -infinite_loop=0 -del_vacant_loop=0 -struct_alloc=1 -nologo

Linker: -noprelink -nodebug -rom=D=R -nomessage -list="$(CONFIGDIR)\$(PROJECTNAME).map"
-show=symbol,reference,xreference -nooptimize
-start=DVECTTBL,DINTTBL/00,PIntPRG/0400,PResetPRG,P,PMY_CRC_CCITT,CMY_CRC_CCITT,P
MY_CRC16,CMY_CRC16,C,C$DSEC,C$BSEC,D/0800,B,R/0FFFFA000,S/0FFFFBE00 -nologo -stack
-output="$(CONFIGDIR)\$(PROJECTNAME).abs" -end
-input="$(CONFIGDIR)\$(PROJECTNAME).abs" -form=stype
-output="$(CONFIGDIR)\$(PROJECTNAME).mot" –exit

2.2.2.3 Best speed

Compiler: -cpu=sh2
-include="$(WORKSPDIR)\..\..\..\SharedTests\ROM","$(WORKSPDIR)\..\..\..\SharedTests"
-object="$(CONFIGDIR)\$(FILELEAF).obj" -speed -noinline -gbr=auto -chgincpath -errorpath
-global_volatile=0 -opt_range=all -infinite_loop=0 -del_vacant_loop=0 -struct_alloc=1 -nologo

Linker: -noprelink -nodebug -rom=D=R -nomessage -list="$(CONFIGDIR)\$(PROJECTNAME).map"
-show=symbol,reference,xreference -optimize=speed
-start=DVECTTBL,DINTTBL/00,PIntPRG/0400,PResetPRG,P,PMY_CRC_CCITT,CMY_CRC_CCITT,P
MY_CRC16,CMY_CRC16,C,C$DSEC,C$BSEC,D/0800,B,R/0FFFFA000,S/0FFFFBE00 -nologo -stack
-output="$(CONFIGDIR)\$(PROJECTNAME).abs" -end
-input="$(CONFIGDIR)\$(PROJECTNAME).abs" -form=stype
-output="$(CONFIGDIR)\$(PROJECTNAME).mot" -exit

2.3 M16C TINY DEVELOPMENT ENVIRONMENT

Development board: RSK-M16C29, 20 MHz external clock.
MCU: M30290FC, configured for 20 MHz internal clock.
Tool chain: Renesas M16C standard tool chain v5.42.00.
In-circuit debugger: E8.

2.3.1 Tool chain optimisation settings

2.3.1.1 No optimisation

Compiler: -c -finfo -dir "$(CONFIGDIR)" -fFP -Wall -WMT -WNC -WNP -WNS -WNUA -WNUF -WNWS -WSAL -WSAW -Wstdout -WUM -WUP –WUV

Linker: -L "nc30lib" -G -MS -O "$(CONFIGDIR)\$(PROJECTNAME).x30" -JOPT

2.3.1.2 Minimal size

Compiler: -c -finfo -dir "$(CONFIGDIR)" -O5 -OR -O5OA -OGJ -OSA -fCE -fD32 -fFP -fNA -fNC -fSA -fUD -Wall -WMT -WNC -WNP -WNS -WNUA -WNUF -WNWS -WSAL –WSAW

Linker: -L "nc30lib" -G -U -W -MS -O "$(CONFIGDIR)\$(PROJECTNAME).x30"

2.3.1.3 Best speed

Compiler: -c -finfo -dir "$(CONFIGDIR)" -O5 -OS -OFFTI -OGJ -OSA -OSTI -OLU=10 -fCE -fD32 -fFP -fNC -fSA -fUD -Wall -WMT -WNC -WNP -WNS -WNUA -WNUF -WNWS

Linker: -L "nc30lib" -G -U -W -MS -O "$(CONFIGDIR)\$(PROJECTNAME).x30" –JOPT

2.4 H8 TINY DEVELOPMENT ENVIRONMENT

Development board: EDK-3687, 18.432 MHz external clock.
MCU: DF33687GFP, configured for 18.432 MHz internal clock.
In-circuit debugger: E8.

2.4.1 Tool chain optimisation settings for Phase 1

Tool chain: Renesas H8C standard tool chain **v6.1.2.0.**

2.4.1.1 No optimisation

Compiler: -cpu=300HN -regparam=3 -define=Release=1,Release=1
-object="$(CONFIGDIR)\$(FILELEAF).obj" -nolist -optimize=0 -chgincpath -nologo

Linker: -noprelink -nodebug -rom=D=R -nomessage -list="$(CONFIGDIR)\$(PROJECTNAME).map"
-nooptimize
-start=PResetPRG,PIntPRG/0400,P,C,C$DSEC,C$BSEC,D,PMY_CRC32,CMY_CRC32/0800,B,R/0E800,S/0EF00 -nologo -stack -output="$(CONFIGDIR)\$(PROJECTNAME).abs" -end
-input="$(CONFIGDIR)\$(PROJECTNAME).abs" -form=stype
-output="$(CONFIGDIR)\$(PROJECTNAME).mot" -exit

2.4.1.2 Minimal size

Compiler: -cpu=300HN -regparam=3 -define=Release=1,Release=1
-object="$(CONFIGDIR)\$(FILELEAF).obj" -nolist -chgincpath -nologo

Linker: -noprelink -nodebug -rom=D=R -nomessage -list="$(CONFIGDIR)\$(PROJECTNAME).map"
-show=symbol,reference,xreference -nooptimize
-start=PResetPRG,PIntPRG/0400,P,C,C$DSEC,C$BSEC,D,PMY_CRC32,CMY_CRC32/0800,B,R/0E800,S/0EF00 -nologo -stack -output="$(CONFIGDIR)\$(PROJECTNAME).abs" -end
-input="$(CONFIGDIR)\$(PROJECTNAME).abs" -form=stype
-output="$(CONFIGDIR)\$(PROJECTNAME).mot" -exit

2.4.1.3 Best speed

Compiler: -cpu=300HN -regparam=3 -define=Release=1,Release=1
-object="$(CONFIGDIR)\$(FILELEAF).obj" -nolist
-speed=register,switch,shift,struct,expression,loop=2,inline -chgincpath -nologo

Linker: -noprelink -nodebug -rom=D=R -nomessage -list="$(CONFIGDIR)\$(PROJECTNAME).map"
-show=symbol,reference,xreference -optimize=speed
-start=PResetPRG,PIntPRG/0400,P,C,C$DSEC,C$BSEC,D,PMY_CRC32,CMY_CRC32/0800,B,R/0E800,S/0EF00 -nologo -stack -output="$(CONFIGDIR)\$(PROJECTNAME).abs" -end
-input="$(CONFIGDIR)\$(PROJECTNAME).abs" -form=stype
-output="$(CONFIGDIR)\$(PROJECTNAME).mot" –exit

2.4.2 Tool chain optimisation settings for Phase 2

Tool chain: Renesas H8C standard tool chain **v6.2.2.0.**

2.4.2.1 No optimisation

Compiler: -cpu=300HN -regparam=3
-include="$(WORKSPDIR)\..\..\..\SharedTests\ROM","$(WORKSPDIR)\..\..\..\SharedTests"
-define=Release=1,Release=1 -object="$(CONFIGDIR)\$(FILELEAF).obj" -nolist -optimize=0
-chgincpath -nologo

Linker: - -noprelink -nodebug -rom=D=R -nomessage
-list="$(CONFIGDIR)\$(PROJECTNAME).map" -show=symbol,reference,xreference -nooptimize
-start=PResetPRG,PIntPRG/0400,P,C,C$DSEC,C$BSEC,D,PMY_CRC32,CMY_CRC32/0800,B,R/0
E800,S/0EF00 -nologo -stack -output="$(CONFIGDIR)\$(PROJECTNAME).abs" -end
-input="$(CONFIGDIR)\$(PROJECTNAME).abs" -form=stype
-output="$(CONFIGDIR)\$(PROJECTNAME).mot" -exit

2.4.2.2 Minimal size

Compiler: -cpu=300HN -regparam=3
-include="$(WORKSPDIR)\..\..\..\SharedTests\ROM","$(WORKSPDIR)\..\..\..\SharedTests"
-define=Release=1,Release=1 -object="$(CONFIGDIR)\$(FILELEAF).obj" -nolist -chgincpath -nologo

Linker: -noprelink -nodebug -rom=D=R -nomessage -list="$(CONFIGDIR)\$(PROJECTNAME).map"
-show=symbol,reference,xreference -nooptimize
-start=PResetPRG,PIntPRG/0400,P,C,C$DSEC,C$BSEC,D,PMY_CRC32,CMY_CRC32/0800,B,R/0
E800,S/0EF00 -nologo -stack -output="$(CONFIGDIR)\$(PROJECTNAME).abs" -end
-input="$(CONFIGDIR)\$(PROJECTNAME).abs" -form=stype
-output="$(CONFIGDIR)\$(PROJECTNAME).mot" -exit

2.4.2.3 Best speed

Compiler: -cpu=300HN -regparam=3
-include="$(WORKSPDIR)\..\..\..\SharedTests\ROM","$(WORKSPDIR)\..\..\..\SharedTests"
-define=Release=1,Release=1 -object="$(CONFIGDIR)\$(FILELEAF).obj" -nolist
-speed=register,switch,shift,struct,expression,loop=2,inline -chgincpath -nologo

Linker: -noprelink -nodebug -rom=D=R -nomessage -list="$(CONFIGDIR)\$(PROJECTNAME).map"
-show=symbol,reference,xreference -optimize=speed
-start=PResetPRG,PIntPRG/0400,P,C,C$DSEC,C$BSEC,D,PMY_CRC32,CMY_CRC32/0800,B,R/0
E800,S/0EF00 -nologo -stack -output="$(CONFIGDIR)\$(PROJECTNAME).abs" -end
-input="$(CONFIGDIR)\$(PROJECTNAME).abs" -form=stype
-output="$(CONFIGDIR)\$(PROJECTNAME).mot" -exit

3 BENCHMARKING RESULTS

The function execution time was measured using the pulse-width measurement function on a TDS3034B digital oscilloscope. A port pin was set low at function entry and high at function exit.

The clock cycle count was calculated using the following equation:

$$Clock\ cycles = f_{CPU} \times t_{FUNCTION}$$

where : f_{CPU} is the CPU clock frequency (Hz)
$t_{FUNCTION}$ is the function execution time (seconds)

For clarity the clock cycle counts have been scaled down by 1000.

3.1 R8C TINY

3.1.1 CPU test results

Table 4: R8C/Tiny CPU test results

Measurement	Result
Code size / bytes	501
Stack usage / bytes	24
Clock cycle count	1214

3.1.2 RAM test results

The tests were executed in 8 and 16-bit configurations.
The non-destructive tests were omitted for 1024 bytes due to insufficient RAM on this device.

3.1.2.1 March C

Table 5: R8C/Tiny March C test results (8-bit)

Measurement			Optimisation		
			None	Size	Speed
Code size / bytes			651	569	815
Stack usage / bytes			56	43	45
Clock cycle count (/ 1000)	Destructive	1024 bytes	4012	3200	3044
		500 bytes	1958	1562	1486
		100 bytes	392	312	298
	Non-destructive	1024 bytes	-	-	-
		500 bytes	2028	1634	1562
		100 bytes	406	328	312
Time Measured (ms)	Destructive	1024 bytes	200.60	160.00	152.20
		500 bytes	97.90	78.10	74.30
		100 bytes	19.60	15.60	14.90
	Non-destructive	1024 bytes	-	-	-
		500 bytes	101.40	81.70	78.10
		100 bytes	20.30	16.40	15.60

Table 6: R8C/Tiny March C test results (16-bit)

Measurement			Optimisation		
			None	Size	Speed
Code size / bytes			711	628	746
Stack usage / bytes			58	43	38
Clock cycle count (/ 1000)	Destructive	1024 bytes	3854	3260	2134
		500 bytes	1882	1592	1044
		100 bytes	376	320	210
	Non-destructive	1024 bytes	-	-	-
		500 bytes	1920	1628	1082
		100 bytes	384	326	218
Time Measured (ms)	Destructive	1024 bytes	192.70	163.00	106.70
		500 bytes	94.10	79.60	52.20
		100 bytes	18.80	16.00	10.50
	Non-destructive	1024 bytes	-	-	-
		500 bytes	96.00	81.40	54.10
		100 bytes	19.20	16.30	10.90

3.1.2.2 March X

Table 7: R8C/Tiny March X test results (8-bit)

Measurement			Optimisation		
			None	Size	Speed
Code size / bytes			589	524	539
Stack usage / bytes			56	43	34
Clock cycle count (/ 1000)	Destructive	1024 bytes	2072	1584	1556
		500 bytes	1012	774	760
		100 bytes	204	154	152
	Non-destructive	1024 bytes	-	-	-
		500 bytes	1080	840	836
		100 bytes	216	168	168
Time Measured (ms)	Destructive	1024 bytes	103.6	79.2	77.8
		500 bytes	50.6	38.7	38.0
		100 bytes	10.2	7.7	7.6
	Non-destructive	1024 bytes	-	-	-
		500 bytes	54.0	42	41.8
		100 bytes	10.8	8.4	8.4

Table 8: R8C/Tiny March X test results (16-bit)

Measurement			Optimisation		
			None	Size	Speed
Code size / bytes			651	586	612
Stack usage / bytes			58	43	35
Clock cycle count (/ 1000)	Destructive	1024 bytes	1960	1662	1632
		500 bytes	958	812	798
		100 bytes	192	162	160
	Non-destructive	1024 bytes	-	-	-
		500 bytes	994	848	836
		100 bytes	200	170	168
Time Measured (ms)	Destructive	1024 bytes	98.0	83.1	81.6
		500 bytes	47.9	40.6	39.9
		100 bytes	9.6	8.10	8.0
	Non-destructive	1024 bytes	-	-	-
		500 bytes	49.7	42.4	41.8
		100 bytes	10.0	8.5	8.4

3.1.2.3 March X WOM

Table 9: R8C/Tiny March X WOM test results (8-bit)

Measurement			Optimisation		
			None	Size	Speed
Code size / bytes			545	470	620
Stack usage / bytes			54	39	40
Clock cycle count (/ 1000)	Destructive	1024 bytes	576	498	510
		500 bytes	282	244	250
		100 bytes	56	50	50
	Non-destructive	1024 bytes	-	-	-
		500 bytes	350	310	322
		100 bytes	70	62	64
Time Measured (ms)	Destructive	1024 bytes	28.8	24.9	25.5
		500 bytes	14.1	12.2	12.5
		100 bytes	2.8	2.5	2.5
	Non-destructive	1024 bytes	-	-	-
		500 bytes	17.5	15.5	16.1
		100 bytes	3.5	3.1	3.2

Table 10: R8C/Tiny March X WOM test results (16-bit)

Measurement			Optimisation		
			None	Size	Speed
Code size / bytes			615	547	715
Stack usage / bytes			57	41	42
Clock cycle count (/ 1000)	Destructive	1024 bytes	310	290	288
		500 bytes	152	142	142
		100 bytes	32	28	28
	Non-destructive	1024 bytes	-	-	-
		500 bytes	188	180	182
		100 bytes	38	36	36
Time Measured (ms)	Destructive	1024 bytes	15.5	14.5	14.4
		500 bytes	7.6	7.1	7.1
		100 bytes	1.6	1.4	1.4
	Non-destructive	1024 bytes	-	-	-
		500 bytes	9.4	9	9.1
		100 bytes	1.9	1.8	1.8

3.1.3 ROM test results

Table 11: R8C/Tiny test results (CRC16-CCITT using a static table)

		Optimisation		
Measurement		**None**	**Size**	**Speed**
Code size / bytes		107	73	73
Constant size / bytes		512	512	512
Stack usage / bytes		21	17	17
Clock cycle count (/ 1000)	1 kbytes	136	104	104
	16 kbytes	2160	1664	1660
	64 kbytes	8200	6160	6160
	128 kbytes	16400	12340	12340
Time Measured (ms)	1 kbytes	6.8	5.2	5.2
	16 kbytes	108	83	83
	64 kbytes	410	308	308
	128 kbytes	820	617	617

Table 12: R8C/Tiny test results (CRC16-CCITT using bit shifting)

		Optimisation		
Measurement		**None**	**Size**	**Speed**
Code size / bytes		129	84	84
Constant size / bytes		0	0	0
Stack usage / bytes		19	9	9
Clock cycle count (/ 1000)	1 kbytes	192	132	124
	16 kbytes	3040	2080	1972
	64 kbytes	11660	7800	7400
	128 kbytes	23400	15600	14820
Time Measured (ms)	1 kbytes	9.6	6.6	6.2
	16 kbytes	152	104	99
	64 kbytes	583	390	370
	128 kbytes	1170	780	741

Table 13: R8C/Tiny test results (CRC16-CCITT with small lookup table)

		Optimisation		
Measurement		**None**	**Size**	**Speed**
Code size / bytes		177	152	152
Constant size / bytes		32	32	32
Stack usage / bytes		15	14	14
Clock cycle count (/ 1000)	1 kbytes	216	224	224
	16 kbytes	3480	3580	3580
	64 kbytes	13520	13840	13840
	128 kbytes	27000	27600	27600
Time Measured (ms)	1 kbytes	10.8	11.2	11.2
	16 kbytes	174	179	179
	64 kbytes	676	692	692
	128 kbytes	1350	1380	1380

Table 14: R8C/Tiny test results (CRC16 with small lookup table)

Measurement		Optimisation		
		None	Size	Speed
Code size / bytes		166	178	144
Constant size / bytes		32	32	32
Stack usage / bytes		15	11	11
Clock cycle count (/ 1000)	1 kbytes	252	224	224
	16 kbytes	4040	3580	3580
	64 kbytes	15640	13840	13840
	128 kbytes	31200	27600	27600
Time Measured (ms)	1 kbytes	12.6	11.2	11.2
	16 kbytes	202	179	179
	64 kbytes	782	692	692
	128 kbytes	1560	1380	1380

3.2 SH TINY

3.2.1 CPU test results

Table 15: SH/Tiny CPU test results

Measurement	Result
Code size / bytes	892
Stack usage / bytes	4
Clock cycle count	304

3.2.2 RAM test results

The tests were executed in 16 and 32-bit configurations.

3.2.2.1 March C

Table 16: SH/Tiny March C test results (16-bit)

Measurement			Optimisation		
			None	Size	Speed
Code size / bytes			830	516	1256
Stack usage / bytes			92	68	72
Clock cycle count (/ 1000)	Destructive	1024 bytes	1717	962	633
		500 bytes	839	470	309
		100 bytes	168	94	62
	Non-destructive	1024 bytes	1731	972	640
		500 bytes	845	474	313
		100 bytes	169	96	63
Time Measured (ms)	Destructive	1024 bytes	42.93	24.06	15.82
		500 bytes	20.97	11.75	7.73
		100 bytes	4.20	2.35	1.55
	Non-destructive	1024 bytes	43.27	24.29	16.01
		500 bytes	21.13	11.86	7.82
		100 bytes	4.23	2.39	1.57

Table 17: SH/Tiny March C test results (32-bit)

Measurement			Optimisation		
			None	Size	Speed
Code size / bytes			836	502	1172
Stack usage / bytes			108	64	68
Clock cycle count (/ 1000)	Destructive	1024 bytes	2290	876	568
		500 bytes	1118	428	278
		100 bytes	224	86	56
	Non-destructive	1024 bytes	2298	880	572
		500 bytes	1122	430	279
		100 bytes	225	86	56
Time Measured (ms)	Destructive	1024 bytes	57.25	21.89	14.21
		500 bytes	27.96	10.69	6.94
		100 bytes	5.60	2.14	1.39
	Non-destructive	1024 bytes	57.44	22.00	14.30
		500 bytes	28.05	10.75	6.98
		100 bytes	5.62	2.16	1.40

3.2.2.2 March X

Table 18: SH/Tiny March X test results (16-bit)

Measurement			Optimisation		
			None	Size	Speed
Code size / bytes			702	474	1308
Stack usage / bytes			88	60	72
Clock cycle count (/ 1000)	Destructive	1024 bytes	748	464	308
		500 bytes	364	228	150
		100 bytes	72	44	30
	Non-destructive	1024 bytes	764	472	313
		500 bytes	372	232	153
		100 bytes	76	48	31
Time Measured (ms)	Destructive	1024 bytes	18.7	11.6	7.70
		500 bytes	9.12	5.70	3.76
		100 bytes	1.80	1.10	0.76
	Non-destructive	1024 bytes	19.1	11.8	7.82
		500 bytes	9.30	5.80	3.82
		100 bytes	1.90	1.20	0.77

Table 19: SH/Tiny March X test results (32-bit)

Measurement			Optimisation		
			None	Size	Speed
Code size / bytes			660	458	1212
Stack usage / bytes			88	56	68
Clock cycle count (/ 1000)	Destructive	1024 bytes	612	428	288
		500 bytes	300	208	140
		100 bytes	60	44	28
	Non-destructive	1024 bytes	620	432	290
		500 bytes	308	212	142
		100 bytes	60	44	29
Time Measured (ms)	Destructive	1024 bytes	15.3	10.7	7.20
		500 bytes	7.5	5.2	3.51
		100 bytes	1.5	1.1	0.7
	Non-destructive	1024 bytes	15.5	10.8	7.26
		500 bytes	7.7	5.3	3.55
		100 bytes	1.5	1.1	0.72

3.2.2.3 March X WOM

Table 20: SH/Tiny March X WOM test results (16-bit)

Measurement			Optimisation None	Size	Speed
Code size / bytes			576	398	654
Stack usage / bytes			72	48	56
Clock cycle count (/ 1000)	Destructive	1024 bytes	64	50	44
		500 bytes	32	25	22
		100 bytes	6	5	4
	Non-destructive	1024 bytes	78	60	49
		500 bytes	38	29	24
		100 bytes	8	6	5
Time Measured (ms)	Destructive	1024 bytes	1.6	1.26	1.1
		500 bytes	0.79	0.62	0.54
		100 bytes	0.16	0.13	0.11
	Non-destructive	1024 bytes	1.94	1.49	1.22
		500 bytes	0.95	0.73	0.6
		100 bytes	0.2	0.15	0.12

Table 21: SH/Tiny March X WOM test results (32-bit)

Measurement			Optimisation None	Size	Speed
Code size / bytes			552	372	604
Stack usage / bytes			68	44	48
Clock cycle count (/ 1000)	Destructive	1024 bytes	32	22	20
		500 bytes	16	11	10
		100 bytes	3	2	2
	Non-destructive	1024 bytes	40	27	23
		500 bytes	20	13	11
		100 bytes	4	3	2
Time Measured (ms)	Destructive	1024 bytes	0.81	0.56	0.51
		500 bytes	0.40	0.28	0.25
		100 bytes	0.08	0.06	0.06
	Non-destructive	1024 bytes	1.00	0.68	0.57
		500 bytes	0.50	0.33	0.28
		100 bytes	0.10	0.07	0.06

3.2.3 ROM test results

Table 22: SH/Tiny test results (CRC16-CCITT using a static table)

Measurement		Optimisation		
		None	Size	Speed
Code size / bytes		164	64	68
Constant size / bytes		512	512	512
Stack usage / bytes		44	4	8
Clock cycle count (/ 1000)	1 kbytes	112	24	24
	16 kbytes	1568	368	312
	64 kbytes	6320	1440	1240
	128 kbytes	12600	2896	2488
Time Measured (ms)	1 kbytes	2.80	0.60	0.60
	16 kbytes	39.20	9.20	7.80
	64 kbytes	158.00	36.00	31.00
	128 kbytes	315.00	72.40	62.20

Table 23: SH/Tiny test results (CRC16-CCITT using bit shifting)

Measurement		Optimisation		
		None	Size	Speed
Code size / bytes		276	78	80
Constant size / bytes		0	0	0
Stack usage / bytes		68	4	4
Clock cycle count (/ 1000)	1 kbytes	272	32	24
	16 kbytes	4360	504	488
	64 kbytes	17440	2032	1960
	128 kbytes	34880	4080	3928
Time Measured (ms)	1 kbytes	6.80	0.80	0.60
	16 kbytes	109.00	12.60	12.20
	64 kbytes	436.00	50.80	49.00
	128 kbytes	872.00	102.00	98.20

Table 24: SH/Tiny test results (CRC16-CCITT with small lookup table)

Measurement		Optimisation		
		None	Size	Speed
Code size / bytes		288	180	180
Constant size / bytes		32	32	32
Stack usage / bytes		44	24	20
Clock cycle count (/ 1000)	1 kbytes	112	64	64
	16 kbytes	1848	1032	984
	64 kbytes	7400	4120	3944
	128 kbytes	14800	8240	7880
Time Measured (ms)	1 kbytes	2.8	1.6	1.6
	16 kbytes	46.2	25.8	24.6
	64 kbytes	185	103	98.6
	128 kbytes	370	206	197

Table 25: SH/Tiny test results (CRC16 with small lookup table)

Measurement		Optimisation		
		None	Size	Speed
Code size / bytes		208	120	128
Constant size / bytes		32	32	32
Stack usage / bytes		32	8	12
Clock cycle count (/ 1000)	1 kbytes	88	48	40
	16 kbytes	1400	752	720
	64 kbytes	5560	3016	2880
	128 kbytes	11120	6040	5760
Time Measured (ms)	1 kbytes	2.2	1.2	1.0
	16 kbytes	35	18.8	18.0
	64 kbytes	139	75.4	72.0
	128 kbytes	278	151	144

3.3 M16C TINY

3.3.1 CPU test results

Table 26: M16C/Tiny CPU test results

Measurement	Result
Code size / bytes	521
Stack usage / bytes	24
Clock cycle count	716

3.3.2 RAM test results

The tests were executed in 8 and 16-bit configurations.

3.3.2.1 March C

Table 27: M16C/Tiny March C test results (8-bit)

Measurement			Optimisation		
			None	Size	Speed
Code size / bytes			664	582	844
Stack usage / bytes			56	43	48
Clock cycle count (/ 1000)	Destructive	1024 bytes	3160	2244	2376
		500 bytes	1544	1144	1160
		100 bytes	308	220	232
	Non-destructive	1024 bytes	3282	2448	2512
		500 bytes	1602	1152	1226
		100 bytes	322	240	246
Time Measured (ms)	Destructive	1024 bytes	158.00	112.20	118.80
		500 bytes	77.20	57.20	58.00
		100 bytes	15.40	11.00	11.60
	Non-destructive	1024 bytes	164.10	122.40	125.60
		500 bytes	80.10	57.60	61.30
		100 bytes	16.10	12.00	12.30

Table 28: M16C/Tiny March C test results (16-bit)

Measurement			Optimisation		
			None	Size	Speed
Code size / bytes			728	641	795
Stack usage / bytes			58	43	42
Clock cycle count (/ 1000)	Destructive	1024 bytes	2912	2340	1780
		500 bytes	1422	1114	870
		100 bytes	284	228	174
	Non-destructive	1024 bytes	2982	2336	1850
		500 bytes	1456	1170	902
		100 bytes	292	228	182
Time Measured (ms)	Destructive	1024 bytes	145.60	117.00	89.00
		500 bytes	71.10	55.70	43.50
		100 bytes	14.20	11.40	8.70
	Non-destructive	1024 bytes	149.10	116.80	92.50
		500 bytes	72.80	58.50	45.10
		100 bytes	14.60	11.40	9.10

3.3.2.2 March X

Table 29: M16C/Tiny March X test results (8-bit)

Measurement			Optimisation		
			None	Size	Speed
Code size / bytes			604	537	568
Stack usage / bytes			56	43	37
Clock cycle count (/ 1000)	Destructive	1024 bytes	1529	1162	1146
		500 bytes	747	568	560
		100 bytes	150	114	112
	Non-destructive	1024 bytes	1640	1273	1266
		500 bytes	801	622	618
		100 bytes	161	125	124
Time Measured (ms)	Destructive	1024 bytes	76.45	58.12	57.30
		500 bytes	37.34	28.38	27.98
		100 bytes	7.49	5.70	5.60
	Non-destructive	1024 bytes	81.98	63.66	63.28
		500 bytes	40.04	31.10	30.90
		100 bytes	8.04	6.24	6.18

Table 30: M16C/Tiny March X test results (16-bit)

Measurement			Optimisation		
			None	Size	Speed
Code size / bytes			668	599	573
Stack usage / bytes			58	43	34
Clock cycle count (/ 1000)	Destructive	1024 bytes	1258	1190	941
		500 bytes	614	581	460
		100 bytes	123	116	92
	Non-destructive	1024 bytes	1324	1247	1011
		500 bytes	646	609	494
		100 bytes	130	122	99
Time Measured (ms)	Destructive	1024 bytes	62.88	59.48	47.06
		500 bytes	30.7	29.04	22.98
		100 bytes	6.16	5.82	4.6
	Non-destructive	1024 bytes	66.18	62.36	50.53
		500 bytes	32.32	30.46	24.68
		100 bytes	6.48	6.12	4.94

3.3.2.3 March X WOM

Table 31: M16C/Tiny March X WOM test results (8-bit)

Measurement			Optimisation		
			None	Size	Speed
Code size / bytes			560	483	650
Stack usage / bytes			54	39	43
Clock cycle count (/ 1000)	Destructive	1024 bytes	359	355	334
		500 bytes	175	174	164
		100 bytes	36	35	33
	Non-destructive	1024 bytes	470	465	455
		500 bytes	230	228	223
		100 bytes	46	46	45
Time Measured (ms)	Destructive	1024 bytes	17.94	17.74	16.7
		500 bytes	8.76	8.68	8.18
		100 bytes	1.78	1.76	1.64
	Non-destructive	1024 bytes	23.48	23.26	22.74
		500 bytes	11.48	11.38	11.14
		100 bytes	2.32	2.3	2.24

Table 32: M16C/Tiny March X WOM test results (16-bit)

Measurement			Optimisation		
			None	Size	Speed
Code size / bytes			632	560	743
Stack usage / bytes			57	41	44
Clock cycle count (/ 1000)	Destructive	1024 bytes	221	188	221
		500 bytes	108	92	108
		100 bytes	22	19	22
	Non-destructive	1024 bytes	287	247	290
		500 bytes	140	121	142
		100 bytes	29	24	29
Time Measured (ms)	Destructive	1024 bytes	11.06	9.42	11.06
		500 bytes	5.42	4.62	5.42
		100 bytes	1.12	0.96	1.10
	Non-destructive	1024 bytes	14.36	12.34	14.48
		500 bytes	7.02	6.04	7.10
		100 bytes	1.44	1.22	1.44

3.3.3 ROM test results

Table 33: M16C/Tiny test results (CRC16-CCITT using a static table)

Measurement		Optimisation		
		None	Size	Speed
Code size / bytes		117	81	81
Constant size / bytes		512	512	512
Stack usage / bytes		23	19	19
Clock cycle count (/ 1000)	1 kbytes	96	80	78
	16 kbytes	1540	1228	1228
	64 kbytes	5832	4656	4652
	128 kbytes	11664	9288	9300
Time Measured (ms)	1 kbytes	4.81	4.00	3.90
	16 kbytes	77.00	61.40	61.40
	64 kbytes	291.60	232.80	232.60
	128 kbytes	583.20	464.40	465.00

Table 34: M16C/Tiny test results (CRC16-CCITT using bit shifting)

Measurement		Optimisation		
		None	Size	Speed
Code size / bytes		136	96	96
Constant size / bytes		0	0	0
Stack usage / bytes		21	19	19
Clock cycle count (/ 1000)	1 kbytes	136	112	103
	16 kbytes	2144	1752	1654
	64 kbytes	8256	6748	6360
	128 kbytes	16516	13500	12712
Time Measured (ms)	1 kbytes	6.80	5.60	5.17
	16 kbytes	107.20	87.60	82.72
	64 kbytes	412.80	337.40	318.00
	128 kbytes	825.80	675.00	635.60

Table 35: M16C/Tiny test results (CRC16-CCITT using the on-chip CRC module)

Measurement		Optimisation		
		None	Size	Speed
Code size / bytes		95	74	74
Constant size / bytes		0	0	0
Stack usage / bytes		19	19	19
Clock cycle count (/ 1000)	1 kbytes	48	48	44
	16 kbytes	770	736	736
	64 kbytes	2818	2684	2684
	128 kbytes	5636	5376	5376
Time Measured (ms)	1 kbytes	2.40	2.41	2.21
	16 kbytes	38.50	36.80	36.80
	64 kbytes	140.90	134.20	134.20
	128 kbytes	281.80	268.80	268.80

Table 36: M16C/Tiny test results (CRC16-CCITT with small lookup table)

Measurement		Optimisation		
		None	Size	Speed
Code size / bytes		200	145	145
Constant size / bytes		32	32	32
Stack usage / bytes		17	14	14
Clock cycle count (/ 1000)	1 kbytes	192	156	156
	16 kbytes	2888	2500	2500
	64 kbytes	11900	9560	9560
	128 kbytes	23800	19140	19120
Time Measured (ms)	1 kbytes	9.6	7.8	7.8
	16 kbytes	144.4	125	125
	64 kbytes	595	478	478
	128 kbytes	1190	957	956

Table 37: M16C/Tiny test results (CRC16 with small lookup table)

Measurement		Optimisation		
		None	Size	Speed
Code size / bytes		188	147	147
Constant size / bytes		32	32	32
Stack usage / bytes		17	17	17
Clock cycle count (/ 1000)	1 kbytes	180	140	140
	16 kbytes	2880	2240	2240
	64 kbytes	11080	8660	8660
	128 kbytes	22200	17320	17320
Time Measured (ms)	1 kbytes	9	7	7
	16 kbytes	144	112	112
	64 kbytes	554	433	433
	128 kbytes	1110	866	866

3.4 H8/TINY

3.4.1 CPU test results

Table 38: H8/Tiny CPU test results

Measurement	Result
Code size / bytes	508
Stack usage / bytes	8
Clock cycle count	752

3.4.2 RAM test results

The tests were executed in 8 and 16-bit configurations.
The non-destructive tests were omitted for 1024 bytes due to insufficient RAM on this device.

3.4.2.1 March C

Table 39: H8/Tiny March C test results (8-bit)

Measurement			Optimisation		
			None	Size	Speed
Code size / bytes			946	400	462
Stack usage / bytes			80	56	42
Clock cycle count (/ 1000)	Destructive	1024 bytes	5019	1602	1602
		500 bytes	2451	783	782
		100 bytes	490	157	157
	Non-destructive	1024 bytes	-	-	-
		500 bytes	2516	807	805
		100 bytes	503	162	162
Time Measured (ms)	Destructive	1024 bytes	272.30	86.90	86.90
		500 bytes	133.00	42.50	42.40
		100 bytes	26.60	8.50	8.50
	Non-destructive	1024 bytes	-	-	-
		500 bytes	136.50	43.80	43.70
		100 bytes	27.30	8.80	8.80

Table 40: H8/Tiny March C test results (16-bit)

Measurement			Optimisation		
			None	Size	Speed
Code size / bytes			990	476	498
Stack usage / bytes			80	66	38
Clock cycle count (/ 1000)	Destructive	1024 bytes	4708	1539	1506
		500 bytes	2300	752	735
		100 bytes	461	151	147
	Non-destructive	1024 bytes	-	-	-
		500 bytes	2333	767	748
		100 bytes	468	155	149
Time Measured (ms)	Destructive	1024 bytes	255.40	83.50	81.70
		500 bytes	124.80	40.80	39.90
		100 bytes	25.00	8.20	8.00
	Non-destructive	1024 bytes	-	-	-
		500 bytes	126.60	41.60	40.60
		100 bytes	25.40	8.40	8.10

3.4.2.2 March X

Table 41: H8/Tiny March X test results (8-bit)

Measurement			Optimisation		
			None	Size	Speed
Code size / bytes			884	364	410
Stack usage / bytes			80	56	34
Clock cycle count (/ 1000)	Destructive	1024 bytes	2547	803	803
		500 bytes	1244	393	392
		100 bytes	249	79	79
	Non-destructive	1024 bytes	-	-	-
		500 bytes	1309	417	417
		100 bytes	263	84	84
Time Measured (ms)	Destructive	1024 bytes	138.20	43.58	43.56
		500 bytes	67.50	21.30	21.28
		100 bytes	13.50	4.28	4.26
	Non-destructive	1024 bytes	-	-	-
		500 bytes	71.02	22.62	22.6
		100 bytes	14.28	4.56	4.54

Table 42: H8/Tiny March X test results (16-bit)

Measurement			Optimisation		
			None	Size	Speed
Code size / bytes			940	444	478
Stack usage / bytes			80	56	38
Clock cycle count (/ 1000)	Destructive	1024 bytes	2381	781	763
		500 bytes	1163	382	373
		100 bytes	234	77	75
	Non-destructive	1024 bytes	-	-	-
		500 bytes	1198	396	388
		100 bytes	241	80	78
Time Measured (ms)	Destructive	1024 bytes	129.20	42.36	41.4
		500 bytes	63.12	20.70	20.24
		100 bytes	12.68	4.18	4.06
	Non-destructive	1024 bytes	-	-	-
		500 bytes	65.00	21.48	21.04
		100 bytes	13.06	4.34	4.24

3.4.2.3 March X WOM

Table 43: H8/Tiny March X WOM test results (8-bit)

Measurement			Optimisation		
			None	Size	Speed
Code size / bytes			604	282	450
Stack usage / bytes			76	56	68
Clock cycle count (/ 1000)	Destructive	1024 bytes	522	187	211
		500 bytes	255	92	103
		100 bytes	52	19	21
	Non-destructive	1024 bytes	-	-	-
		500 bytes	320	117	127
		100 bytes	65	24	26
Time Measured (ms)	Destructive	1024 bytes	28.30	10.16	11.46
		500 bytes	13.84	5.00	5.60
		100 bytes	2.84	1.04	1.14
	Non-destructive	1024 bytes	-	-	-
		500 bytes	17.34	6.32	6.90
		100 bytes	3.54	1.32	1.40

Table 44: H8/Tiny March X WOM test results (16-bit)

Measurement			Optimisation		
			None	Size	Speed
Code size / bytes			770	362	514
Stack usage / bytes			76	56	64
Clock cycle count (/ 1000)	Destructive	1024 bytes	273	97	110
		500 bytes	134	48	54
		100 bytes	28	10	11
	Non-destructive	1024 bytes	-	-	-
		500 bytes	169	62	68
		100 bytes	35	13	14
Time Measured (ms)	Destructive	1024 bytes	14.80	5.28	5.98
		500 bytes	7.28	2.60	2.92
		100 bytes	1.50	0.56	0.60
	Non-destructive	1024 bytes	-	-	-
		500 bytes	9.14	3.38	3.68
		100 bytes	1.92	0.72	0.76

3.4.3 ROM test results

The block count was capped at 56k bytes as this is the maximum supported by the H8/3687 group.

Table 45: H8/Tiny test results (CRC16-CCITT using a static table)

Measurement		Optimisation		
		None	Size	Speed
Code size / bytes		144	56	70
Constant size / bytes		512	512	512
Stack usage / bytes		40	32	18
Clock cycle count (/ 1000)	1 kbytes	111	37	37
	16 kbytes	1769	627	619
	56 kbytes	6193	2182	2182
Time Measured (ms)	1 kbytes	6.01	2.01	2.01
	16 kbytes	96.00	34.01	33.61
	56 kbytes	336.00	118.40	118.40

Table 46: H8/Tiny test results (CRC16-CCITT using bit shifting)

Measurement		Optimisation		
		None	Size	Speed
Code size / bytes		202	88	106
Constant size / bytes		0	0	0
Stack usage / bytes		38	32	16
Clock cycle count (/ 1000)	1 kbytes	229	111	81
	16 kbytes	3635	1770	1246
	56 kbytes	12718	6193	4357
Time Measured (ms)	1 kbytes	12.41	6.04	4.42
	16 kbytes	197.20	96.01	67.62
	56 kbytes	690.00	336.00	236.40

Table 47: H8/Tiny test results (CRC16-CCITT with small lookup table)

Measurement		Optimisation		
		None	Size	Speed
Code size / bytes		334	170	198
Constant size / bytes		32	32	32
Stack usage / bytes		32	20	12
Clock cycle count (/ 1000)	1 kbytes	446	236	166
	16 kbytes	7133	3797	2654
	56 kbytes	25068	13308	9290
Time Measured (ms)	1 kbytes	24.2	12.8	9.0
	16 kbytes	387	206	144
	56 kbytes	1360	722	504

Table 48: H8/Tiny test results (CRC16 with small lookup table)

Measurement		Optimisation		
		None	Size	Speed
Code size / bytes		248	136	150
Constant size / bytes		32	32	32
Stack usage / bytes		28	20	12
Clock cycle count (/ 1000)	1 kbytes	258	133	129
	16 kbytes	2064	2101	2101
	56 kbytes	14450	7336	7336
Time Measured (ms)	1 kbytes	14	7.2	7.0
	16 kbytes	112	114	114
	56 kbytes	784	398	398

3.5 Results Comparison Tables

The following tables compare graphically the RAM and ROM test results for each MCU family.

Table 49: RAM test result comparison

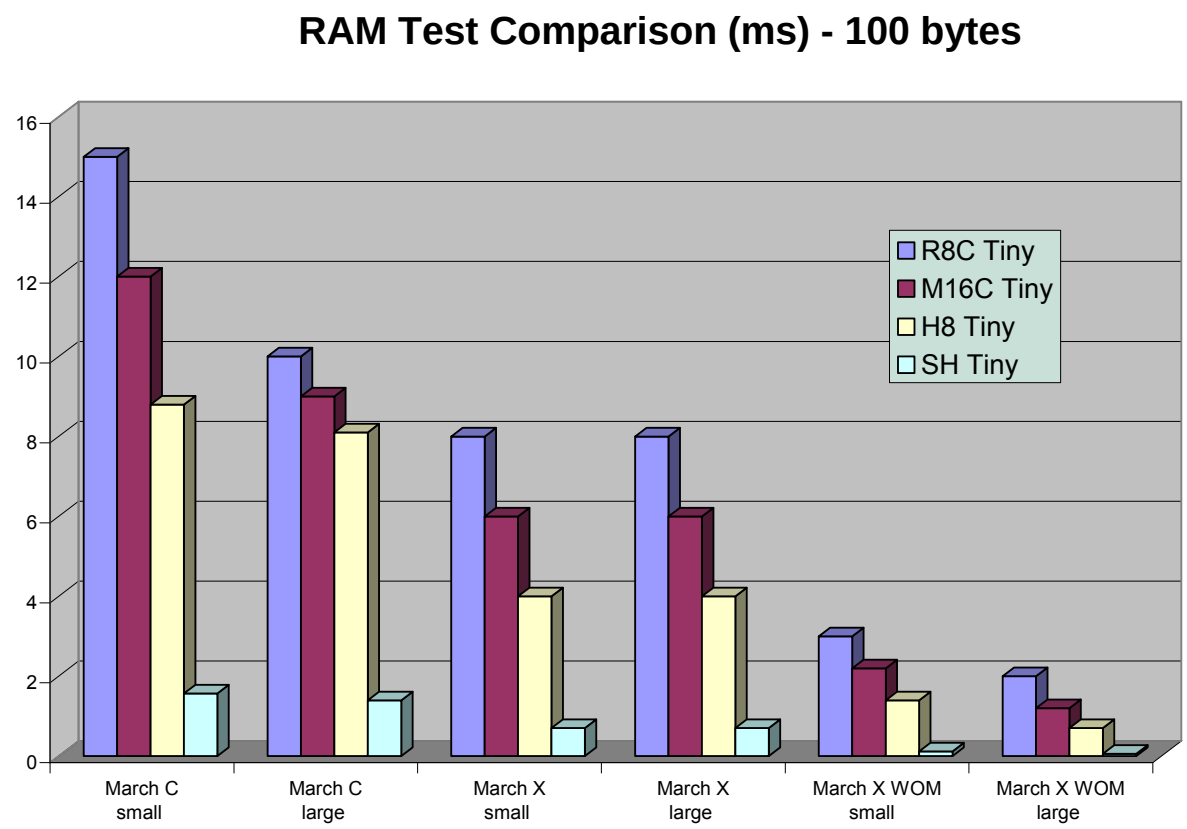
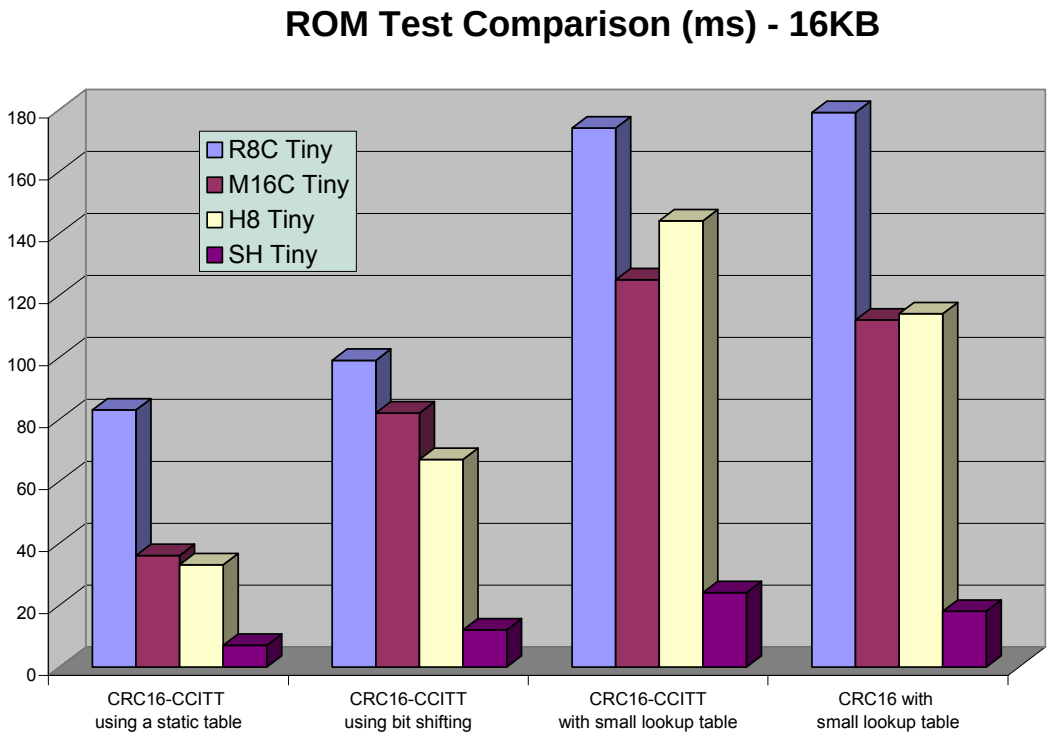


Table 50: ROM test result comparison



4 ABBREVIATIONS

API	Application Programming Interface
CCITT	Comité Consultatif International Téléphonique et Télégraphique, an organisation that sets international communications standards.
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
IEC60730	International Electronics Commission 60730 safety standards
MCU	Micro Controller Unit
MISRA	Motor Industry Software Reliability Association
RAM	Random Access Memory
ROM	Read-Only Memory
WOM	Word Oriented Memory
XOR	Exclusive-OR

5 WEBSITE AND SUPPORT

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

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

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

6 CAUTIONS

Notes regarding these materials

1. This document is provided for reference purposes only so that Renesas customers may select the appropriate Renesas products for their use. Renesas neither makes warranties or representations with respect to the accuracy or completeness of the information contained in this document nor grants any license to any intellectual property rights or any other rights of Renesas or any third party with respect to the information in this document.
2. Renesas shall have no liability for damages or infringement of any intellectual property or other rights arising out of the use of any information in this document, including, but not limited to, product data, diagrams, charts, programs, algorithms, and application circuit examples.
3. You should not use the products or the technology described in this document for the purpose of military applications such as the development of weapons of mass destruction or for the purpose of any other military use. When exporting the products or technology described herein, you should follow the applicable export control laws and regulations, and procedures required by such laws and regulations.
4. All information included in this document such as product data, diagrams, charts, programs, algorithms, and application circuit examples, is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas products listed in this document, please confirm the latest product information with a Renesas sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas such as that disclosed through our website. (http://www.renesas.com)
5. Renesas has used reasonable care in compiling the information included in this document, but Renesas assumes no liability whatsoever for any damages incurred as a result of errors or omissions in the information included in this document.
6. When using or otherwise relying on the information in this document, you should evaluate the information in light of the total system before deciding about the applicability of such information to the intended application. Renesas makes no representations, warranties or guaranties regarding the suitability of its products for any particular application and specifically disclaims any liability arising out of the application and use of the information in this document or Renesas products.
7. With the exception of products specified by Renesas as suitable for automobile applications, Renesas products are not designed, manufactured or tested for applications or otherwise in systems the failure or malfunction of which may cause a direct threat to human life or create a risk of human injury or which require especially high quality and reliability such as safety systems, or equipment or systems for transportation and traffic, healthcare, combustion control, aerospace and aeronautics, nuclear power, or undersea communication transmission. If you are considering the use of our products for such purposes, please contact a Renesas sales office beforehand. Renesas shall have no liability for damages arising out of the uses set forth above.
8. Notwithstanding the preceding paragraph, you should not use Renesas products for the purposes listed below:
   (1) artificial life support devices or systems
   (2) surgical implantations
   (3) healthcare intervention (e.g., excision, administration of medication, etc.)
   (4) any other purposes that pose a direct threat to human life

   Renesas shall have no liability for damages arising out of the uses set forth in the above and purchasers who elect to use Renesas products in any of the foregoing applications shall indemnify and hold harmless Renesas Technology Corp., its affiliated companies and their officers, directors, and employees against any and all damages arising out of such applications.
9. You should use the products described herein within the range specified by Renesas, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas shall have no liability for malfunctions or damages arising out of the use of Renesas products beyond such specified ranges.
10. Although Renesas endeavors to improve the quality and reliability of its products, IC products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Please be sure to implement safety measures to guard against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas 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 applicable measures. Among others, since the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
11. In case Renesas products listed in this document are detached from the products to which the Renesas products are attached or affixed, the risk of accident such as swallowing by infants and small children is very high. You should implement safety measures so that Renesas products may not be easily detached from your products. Renesas shall have no liability for damages arising out of such detachment.
12. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written approval from Renesas.
13. Please contact a Renesas sales office if you have any questions regarding the information contained in this document, Renesas semiconductor products, or if you have any other inquiries.

© 2008. Renesas Technology Corp., All rights reserved.