Preface

IEC 61508 is an international standard governing a range of electrical/electronic/programmable electronic safety-related systems. It defines the requirements needed to ensure that systems are designed, implemented, operated and maintained at the required Safety Integrity Level (SIL). Four SIL levels have been defined to indicate the risks involved in any system, with SIL4 being the highest risk level.
At the heart of many safety- related systems nowadays is a sophisticated and often highly integrated Microcontroller (MCU). An integral part of meeting the requirements of IEC61508 is the ability to verify the correct operation of critical areas of the MCU.
The Renesas Diagnostics Software is designed for use with Renesas' RX 32-bit Microcontroller Family. Tests are provided for coverage for three critical areas of the MCU's operation; The Central Processing Unit (CPU), internal Flash ROM / RAM.

Target device

[RX700 Series]
- RX71M Group
- RX72T Group
- RX72M Group
- RX72N Group

[RX600 Series]
- RX64M Group
- RX65N Group
- RX651 Group
- RX66T Group
- RX66N Group
- RX660 Group
- RX671 Group

[RX200 Series]
- RX230 Group
- RX231 Group
- RX23E-A Group
- RX23T Group
- RX23W Group
- RX24T Group
- RX24U Group

Contents

1. Common Terminology

This chapter defines some common terms and acronyms used throughout the rest of the document and provides references to other Renesas complimentary Documentation.

1.1 Acronyms

Table 1.1 Terminology and acronyms

Acronyms	Description
API	Application Programming Interface
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
DSP	Digital Signal Processor
ECC	Error Correction Code
FPU	Floating Point Unit
HW	Hardware
LUT	Look Up Table
MCU	Micro Controller Unit
MD5	Message Digest Algorithm 5
N/A	Not available
RAM	Random Access Memory
ROM	Read Only Memory
SIL	Safety Integrity Level
SW	Software
TS_ID	Test Segment Identifier
WD	Watch Dog

1.2 Document References

[1] INTERNATIONAL STANDARD IEC61508-1~7 Edition 2.0
[2] RX(v2/v3 core) Family Diagnostic Software Safety Manual Rev.3.00
[3] e2studio 2020-07, e2studio v7.8 User's Manual: Getting Started Guide Rev.1.00
[4] CC-RX Compiler User's Manual Rev.1.11
[5] RX Family RXv3 Instruction Set Architecture User's Manual: Software Rev.1.00
[6] RX Family RXv2 Instruction Set Architecture User's Manual: Software Rev.1.00
[7] RX71M Group User's Manual: Hardware Rev.1.10
[8] RX71M Group User's Manual: Hardware Rev.1.11
[9] RX72T Group User's Manual: Hardware Rev.1.00
[10] RX72N Group User's Manual: Hardware Rev.1.11
[11] RX64M Group User's Manual: Hardware Rev.1.10
[12] RX65N Group, RX651Group User's Manual: Hardware Rev.2.30
[13] RX66TGroup User's Manual: Hardware Rev.1.21
[14] RX66N Group User's Manual: Hardware Rev.1.11
[15] RX660 Group User's Manual: Hardware Rev.1.00
[16] RX671 Group User's Manual: Hardware Rev.1.10
[17] RX230 Group, RX231 Group User's Manual: Hardware Rev.1.20
[18] RX23E-A Group User's Manual: Hardware Rev.1.20
[19] RX23T Group User's Manual: Hardware Rev.1.10
[20] RX23W Group User's Manual: Hardware Rev.1.10
[21] RX24T Group User's Manual: Hardware Rev.2.00
[22] RX24U Group User's Manual: Hardware Rev.1.00

2. Development Environment

The Diagnostic Software code was developed using the following development tools.

Table 2.1 Development tools

Name	Version
e2studio	2022-07
CC-RX	3.01.00

Please refer to [3] when using e2studio.
Please refer to [4] when using CC-RX.

2.1 Type variable

All variables are 32-bit implementations, so care shall be taken that int type must be represented in 32-bit format on the target environment.

2.2 Environment Settings

The e2studio environment should be set up as specified in Table 2.2. The settings are accessible through the [Project] menu > [C/C++ Project Settings].

Table 2.2 e2studio settings

#	Category	Sub-category	Setting description
1	Compiler	Source -> Advanced	C source file (-lang): = C99
2	Compiler	Optimization	Optimize level (-optimize) = 0
3	Compiler	Optimization -> Advanced	Optimizes accesses to external variables (-nomap/-smap/-map): No Perform inter-module optimization (-whole_program/-merge_files/-ip_optimize): No
4	Linker	Section	Refer to 3.3.1.2, 4.4.1.2 for details.

In addition, consider a typical setting for the stack pointers as defined by the following directive.

- *#pragma stacksize su=0x300*
- *#pragma stacksize si=0x100*

Take care that the stack size shall be greater than the one specified in the Resources Usage section.

3. CPU Core Test

3.1 Test Objectives

The objective of the CPU Software Test is to test the correct functionality of the CPU adopting mainly an instruction based diagnosis with the aim to detect permanent hardware failures of the CPU Core.
All instructions, except for the WAIT instruction, is used in the CPU Core Test.
See reference documents [5] [6] for the complete list of instructions. However, please note the primary aim is not to test individual instructions but to detect a hardware failure of the CPU core.

3.2 Software Structure

This software comprises of two different levels of function calls.

a. The first level is the user interface functions named R_CPU_DiagNorm, R_CPU_DiagTbl, and R_CPU_DiagSpec.
b. The second lower level is R_CPU_DiagN functions called by R_CPU_DiagNorm, R_CPU_DiagTbl, R_CPU_DiagSpec functions (Hereinafter simply called as R_CPU_Diag functions).

The R_CPU_Diag function executes the actual diagnostic of the CPU core, whilst the R_CPU_DiagNorm, R_CPU_DiagTbl, and R_CPU_DiagSpec functions allow the user to select and run of one or more of the R_CPU_Diag function(s) in sequence and to collect the diagnostic results.
Up to 71 R_CPU_DiagN functions (N: 0 - 41, 50 - 73, 100 - 104) are available. Table 3.1 and Table 3.2 provide an overview of R_CPU_Diag functions.

There are three types of R_CPU_Diag functions.

- R_CPU_DiagN of type "Fixed" (N: 0 - 41)
  - Operand data necessary to stimulate the core and run these functions is embedded in the code.
  - alled by the R_CPU_DiagNorm function.
- R_CPU_DiagN of type "LUT" (N: 50 - 73)
  - These functions are called with operand data taken from a Look Up Table.
  - Called by the R_CPU_DiagTbl function.
- R_CPU_DiagN of type "Specific" (N: 100 - 104)
  - These functions perform diagnostics in interrupt and exception processing.
  - Called by the R_CPU_DiagSpec function.

Table 3.1 and Table 3.2 also represent previous TS_ID to show relationship with testSegment used in the RX Family Diagnostic Software Ver.1.00. Please note that numbering of Test # has no connection with TS_ID from Ver.1.00.
Some R_CPU_Diag function of type LUT are called with different operand data (Data Set) taken from Look Up Table. The number of Data Set to be used vary depends on R_CPU_Diag function to be called. See "Data Set Number" column in Table 3.2. Note that some functions are not executed depending on the MCU group (as shown below).

Legend

- Supports all MCU groups
- Supports MCU group supporting collective saving of register value
- Supports MCU group with double precision floating point co-processor
- Supports MCU group with software interrupt 2 activation register (SWINT2R)

Table 3.1 R_CPU_Diag Function Overview (1 of 2)

Test #	Function Name	Objective of the Test	Function Type	Data Set Number	Ver.1.00 TS_ID
0	R_CPU_Diag0	Saturate instructions	Fixed	-	2
1	R_CPU_Diag1	Shift, Rotate and TST instructions	Fixed	-	6
2	R_CPU_Diag2	BMCnd instructions	Fixed	-	7
3	R_CPU_Diag3	BCnd instructions	Fixed	-	8
4	R_CPU_Diag4	SCCnd instructions	Fixed	-	9
5	R_CPU_Diag5	BCnd instructions	Fixed	-	10
6	R_CPU_Diag6	BCnd instructions and unconditional jump	Fixed	-	11
7	R_CPU_Diag7	Bit manipulation instructions	Fixed	-	12
8	R_CPU_Diag8	Data transfer instructions	Fixed	-	13
9	R_CPU_Diag9	Instruction queue operations	Fixed	-	30
10	R_CPU_Diag10	LI flag related instructions (MOVCO and MOVLI instructions)	Fixed	-	41
11	R_CPU_Diag11	FTOU instructions	Fixed	-	44
12	R_CPU_Diag12	Integer pipeline operations	Fixed	-	46
13	R_CPU_Diag13	Floating point pipeline operations	Fixed	-	47
14	R_CPU_Diag14	Load/store pipeline operations	Fixed	-	48
15	R_CPU_Diag15	DSP instructions using ACC1 register	Fixed	-	52
16	R_CPU_Diag16	Operand combinations of BCnd and BTST	Fixed	-	54
17	R_CPU_Diag17	General purpose register operations 1st	Fixed	-	29A
18	R_CPU_Diag18	General purpose register operations 2nd	Fixed	-	29B
19	R_CPU_Diag19	Floating point instructions	Fixed	-	31B
20	R_CPU_Diag20	String instructions 1st	Fixed	-	32A
21	R_CPU_Diag21	String instructions 2nd	Fixed	-	32B
22	R_CPU_Diag22	String instructions 3rd	Fixed	-	32C
23	R_CPU_Diag23	Bit manipulation, control registers and data access operations	Fixed	-	34A
24	R_CPU_Diag24	Data access operations	Fixed	-	34B
25	R_CPU_Diag25	Extended DSP Instructions 1st	Fixed	-	42A
26	R_CPU_Diag26	Extended DSP Instructions 2nd	Fixed	-	42B
27	R_CPU_Diag27	Enhanced pipeline opeartions	Fixed	-	New
28	R_CPU_Diag28.	Control register operations, PUSHC and POPC instructions	Fixed	-	17
29	R_CPU_Diag29	General purpose register operations	Fixed	-	33
30	R_CPU_Diag30	Internal pipeline operations 1st	Fixed	-	35A
31	R_CPU_Diag31	Internal pipeline operations 2nd	Fixed	-	35B
32	R_CPU_Diag32	Control register operations	Fixed	-	36

Table 3.2 R_CPU_Diag Function Overview (2 of 2)

Test #	Function Name	Objective of the Test	Function Type	Data Set Number	Ver.1.00 TS_ID
33	R_CPU_Diag33	SAVE and RSTR instruction (Register banks accumulation, R1 to R10)	Fixed	-	New
34	R_CPU_Diag34	SAVE and RSTR instruction (Register banks accumulation, R11 to R15, USP and FPSW)	Fixed	-	New
35	R_CPU_Diag35	SAVE and RSTR instruction (Register banks accumulation, ACC0 and ACC1)	Fixed	-	New
36	R_CPU_Diag36	DMOV instruction (Double Precision FPU)	Fixed	-	New
37	R_CPU_Diag37	DPUSHM.D and DPOPM.D instructions (Double Precision FPU)	Fixed	-	New
38	R_CPU_Diag38	MVTDC, MVFDC and MVFDR instructions (Double Precision FPU)	Fixed	-	New
39	R_CPU_Diag39	DPUSHM.L and DPOPM.L instructions (Double Precision FPU)	Fixed	-	New
40	R_CPU_Diag40	Arithmetic operations (Double Precision FPU)	Fixed	-	New
41	R_CPU_Diag41	Data type conversion (Double Precision FPU)	Fixed	-	New
50	R_CPU_Diag50	Integer addition with carry instructions	LUT	6	23A
51	R_CPU_Diag51	Integer addition without carry instructions	LUT	6	23B
52	R_CPU_Diag52	Integer subtraction with borrow instructions	LUT	6	23C
53	R_CPU_Diag53	Integer subtraction without borrow instructions	LUT	6	23D
54	R_CPU_Diag54	Extended integer signed multiplication instructions	LUT	6	24A
55	R_CPU_Diag55	Extended integer unsigned multiplication instructions	LUT	6	24B
56	R_CPU_Diag56	Integer multiplication instructions	LUT	6	24C
57	R_CPU_Diag57	Integer signed division instructions	LUT	5	25A
58	R_CPU_Diag58	Integer unsigned division instructions	LUT	5	25B
59	R_CPU_Diag59	Floating point addition instructions	LUT	16	26A
60	R_CPU_Diag60	Floating point subtraction instructions	LUT	16	26B
61	R_CPU_Diag61	Floating point multiplication instructions	LUT	16	27
62	R_CPU_Diag62	Floating point division instructions	LUT	16	28
63	R_CPU_Diag63	Integer division instructions	LUT	1	37
64	R_CPU_Diag64	Unsigned integer division instructions	LUT	1	38
65	R_CPU_Diag65	Floating point square root instructions	LUT	10	58
66	R_CPU_Diag66	FSQRT instruction with variable FPSW setting	LUT	2	60
67	R_CPU_Diag67	FSQRT instruction with sequential input	LUT	8	New
68	R_CPU_Diag68	FADD and FSUB instructions with sequential input	LUT	14	New
69	R_CPU_Diag69	FMUL and FDIV instructions with sequential input	LUT	10	New
70	R_CPU_Diag70	ADD and SUB instructions with sequential input	LUT	4	New
71	R_CPU_Diag71	MUL and DIV instructions with sequential input	LUT	6	New
72	R_CPU_Diag72	SAVE and RSTR instruction (Register banks iteration, R1 to R10, USP and FPSW)	LUT	15	New
73	R_CPU_Diag73	SAVE and RSTR instruction (Register banks iteration, R11 to R15, ACC0 and FPSW)	LUT	15	New
100	R_CPU_Diag100	Software interrupt and BRK instruction	Specific	-	59
101	R_CPU_Diag101	Access exception	Specific	-	4A
102	R_CPU_Diag102	Privileged exception	Specific	-	4B
103	R_CPU_Diag103	Software interrupt and undefined instructions exception	Specific	-	14
104	R_CPU_Diag104	Floating point exceptions	Specific	-	18

3.2.1 API and CPU Core Test Environment

The R_CPU_Diag functions of type Fixed are called through the R_CPU_DiagNorm function, of type LUT through the R_CPU_DiagTbl function, and of type Specific through the R_CPU_DiagSpec function.

3.2.1.1 R_CPU_DiagNorm Function

The R_CPU_DiagNorm function prototype is defined as follows:

void R_CPU_DiagNorm(uint32_t index, const uint32_t forceFail, int32_t signatureVector, int32_t* result);*

Table 3.3 describes the input and output of each R_CPU_DiagNorm function.
The parameter forceFail forces the function to fail, by intentionally generating an incorrect signature. This type of software fault injection feature allows for testing of higher level fault handling mechanisms, specified at the application level.

Table 3.3 R_CPU_DiagNorm Function Interface

Type	C Type	Parameter	Description
input	uint32_t	index	Specify the identifier (i.e. index) of which R_CPU_Diag function to run.
input	const uint32_t	forceFail	When set to 0, the function fails forcibly by generating a wrong signature value that is the inverted value of the expected one. All other values do not have any effect on the function behavior.
output	int32_t*	signatureVector	Start address where results of signature values returned from all called R_CPU_Diag functions are stored (Maximum size is 2*32 bits). See Table 3.4 for relationship between index and signature value. See 6.1, Appendix A for expected signature values.
output	int32_t*	result	Pass/fail result of R_CPU_Diag function call. 0 : Failed or invalid index value. 1 : Passed

Table 3.4 shows relationship between identifier (i.e. index) and behavior of R_CPU_Diag function and describes results of function executions.
Note that identifier that can be input differ depending on the MCU group.

Table 3.4 Relationship between Index and R_CPU_Diag Functions including Results of Function Execution

#	Identifier (index)	Description	signatureVector description
1	RX700 Series - RX71M : 0 - 32 - RX72T : 0 - 35 - RX72M : 0 - 41 - RX72N : 0 - 41 RX600 Series - RX66N : 0 - 41 - RX660 : 0 - 35 - RX671 : 0 - 41 - Other RX600 Series : 0 - 32 RX200 Series : 0 - 32	Calls the specified R_CPU_Diag function (N = index number)	signatureVector[0] = signature of called R_CPU_Diag function signatureVector[1] = sign-inverted signature of called R_CPU_Diag function
2	Other index numbers	Function returns a signature value different from the expected value. (signatureVector is set to 0)	signatureVector[0] = 0x00000000 signatureVector[1] = 0x00000000

3.2.1.2 R_CPU_DiagTbl Function

The R_CPU_DiagTbl function prototype is defined as follows:

void R_CPU_DiagTbl(uint32_t index, uint32_t dataSet, const uint32_t forceFail, int32_t signatureVector, int32_t* result);*

Table 3.5 describes the input and output of each R_CPU_DiagTbl function.

Table 3.5 R_CPU_DiagTbl Function Interface

Type	C Type	Parameter	Description
input	uint32_t	Index	Specify the identifier (i.e. index) of which R_CPU_Diag function to run.
input	uint32_t	Dataset	Set of data operand to be used when executing function of type R_CPU_Diag. See Table 3.7 for valid values for each function. If invalid values are entered, then the function is called as dataSet = 0.
input	const uint32_t	forceFail	When set to 0, the function fails forcibly by generating a wrong signature value that is the inverted value of the expected one. All other values do not have any effect on the function behavior.
output	int32_t*	signatureVector	Start address where results of signature values returned from all called R_CPU_Diag functions are stored (Maximum size is 2*32 bits). See Table 3.6 for relationship between index and signature value. See 6.1, Appendix A for expected signature values.
output	int32_t*	result	Pass/fail result of R_CPU_Diag function call. 0 : Failed or invalid index value. 1 : Passed

Table 3.6 shows relationship between identifier (i.e. index) and behavior of R_CPU_Diag function and describes results of function executions.
Note that identifier that can be input differ depending on the MCU group.

Table 3.6 Relationship between Index and R_CPU_Diag Functions including Results of Function Execution

#	Identifier (index)	Description	signatureVector description
1	RX700 Series - RX71M : 0 - 21 - RX72T : 0 - 23 - RX72M : 0 - 23 - RX72N : 0 - 23 RX600 Series - RX66N : 0 - 23 - RX660 : 0 - 23 - RX671 : 0 - 23 - Other RX600 Series : 0 - 21 RX200 Series : 0 - 21	Calls the specified R_CPU_Diag function (N = index + 50)	[0 - 21] signatureVector[0] = signature of called R_CPU_Diag function signatureVector[1] = sign-inverted signature of called R_CPU_Diag function [22 - 23] signatureVector[0] = signature of called R_CPU_Diag function signatureVector[1] = bit-inverted signature of called R_CPU_Diag function
2	Other index numbers	Function returns a signature value different from the expected value. (signatureVector is set to 0)	signatureVector[0] = 0x00000000 signatureVector[1] = 0x00000000

Table 3.7 shows valid values for dataSet parameter.

Table 3.7 Valid dataSet Values

Function name	Valid dataSet values
R_CPU_Diag50	0, 1, 2, 3, 4, 5
R_CPU_Diag51	0, 1, 2, 3, 4, 5
R_CPU_Diag52	0, 1, 2, 3, 4, 5
R_CPU_Diag53	0, 1, 2, 3, 4, 5
R_CPU_Diag54	0, 1, 2, 3, 4, 5
R_CPU_Diag55	0, 1, 2, 3, 4, 5
R_CPU_Diag56	0, 1, 2, 3, 4, 5
R_CPU_Diag57	0, 1, 2, 3, 4
R_CPU_Diag58	0, 1, 2, 3, 4
R_CPU_Diag59	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
R_CPU_Diag60	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
R_CPU_Diag61	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
R_CPU_Diag62	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
R_CPU_Diag63	0
R_CPU_Diag64	0
R_CPU_Diag65	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
R_CPU_Diag66	0, 1
R_CPU_Diag67	0, 1, 2, 3, 4, 5, 6, 7
R_CPU_Diag68	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
R_CPU_Diag69	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
R_CPU_Diag70	0, 1, 2, 3
R_CPU_Diag71	0, 1, 2, 3, 4, 5
R_CPU_Diag72 (RX72T, RX72M, RX72N, RX66N, RX660, RX671)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14
R_CPU_Diag73 (RX72T, RX72M, RX72N, RX66N, RX660, RX671)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14

3.2.1.3 R_CPU_DiagSpec Function

The R_CPU_DiagSpec function prototype is defined as follows:

void R_CPU_DiagSpec(uint32_t index, const uint32_t forceFail, int32_t signatureVector, int32_t* result);*

Table 3.8 describes the input and output of each R_CPU_DiagSpec function.

Table 3.8 R_CPU_DiagSpec Function Interface

Type	C Type	Parameter	Description
input	uint32_t	index	Specify the identifier (i.e. index) of which R_CPU_Diag function to run.
input	const uint32_t	forceFail	When set to 0, the function fails forcibly by generating a wrong signature value that is the inverted value of the expected one. All other values do not have any effect on the function behavior.
output	int32_t*	signatureVector	Start address where results of signature values returned from all called R_CPU_Diag functions are stored (Maximum size is 2*32 bits). See Table 3.9 for relationship between index and signature value. See 6.1, Appendix A for expected signature values.
output	int32_t*	result	Pass/fail result of R_CPU_Diag function call. 0 : Failed or invalid index value. 1 : Passed

Table 3.9 shows relationship between identifier (i.e. index) and behavior of R_CPU_Diag function and describes results of function executions.

Table 3.9 Relationship between Index and R_CPU_Diag Functions including Results of Function Execution

#	Identifier (index)	Description	signatureVector description
1	RX700 Series : 0 - 4 RX600 Series : 0 - 4 RX200 Series : 1 - 4	Calls the specified R_CPU_Diag function (N = index + 100)	signatureVector[0] = signature of called R_CPU_Diag function signatureVector[1] = sign-inverted signature of called R_CPU_Diag function
2	Other index numbers	Function returns a signature value different from the expected value. (signatureVector is set to 0)	signatureVector[0] = 0x00000000 signatureVector[1] = 0x00000000

3.3 Software Integration Rules

This section provides guidelines for how to integrate the CPU Test software within the user project.

3.3.1 Code Integration

3.3.1.1 API Function Call

Follow the instructions below to call the R_CPU_DiagNorm, R_CPU_DiagTbl, and R_CPU_DiagSpec functions (hereinafter simply called as "Test API Function").

1. Deploy this software to directory you want.
2. Add directory listed in Table 3.10 via compiler and assembler setting options.
3. Describe the device type and endians in the testCPU_config.h and testCPU_config.inc.
4. Include the testCPU.h
5. Define input parameters for each function.
    a. *uint32_t index*
    b. *uint32_t dataSet (For R_CPU_DiagTbl only)*
    c. *uint32_t forceFail*
6. Create an int32_t vector of two elements to store the signature values. Then the start address of the vector is passed to the Test API Function.
7. Create an int32_t variable to store the result of the test as "result". Then the address of the variable is passed to Test API Function.

Table 3.10 Additional Directory to Include

Type	Directory
Compiler	/
	/CPU_Test
	/CPU_Test/diag
	/CPU_Test/inc
	/CPU_Test/int_if
	/CPU_Test/rom_if
Assembler	/
	/CPU_Test/inc

NOTE: "/" indicates a directory where this software is deployed.

Example

```
#include "testCPU.h"
.
.
.
    uint32_t index = 7;
    uint32_t dataSet = 0;
    int32_t result = 0;
    int32_t forceFail = 11;
    int32_t signatureVector [2];

    R_CPU_DiagTbl(index, dataSet, forceFail, signatureVector, &result);

    if ((result != 1) || (signatureVector[0] != 0x98519BF7) || (signatureVector[1] != 0x67AE6409)) {
        errorHandler() ;/*Fault detection*/
    }
```

After Test API Function returns, fault can be detected by checking the result output value as shown in the example above then comparing returned signature value to its expected value (See Table 6.1 - Table 6.9 in 6.1, Appendix A). Note that the use of a signature strategy is mandatory in addition to the result variable checking.

A complete example of the Test API Function, which calls all R_CPU_Diag functions is provided in 6.3, Appendix C.

3.3.1.2 Testing of Interrupts and Software Exceptions

To test the interrupt and exception handler, preset vector table (defined in the r_cpu_diag_rx.c, see design file list in 3.5) is used. Place the following sections to the memory via linker setting.

< Sections to be allocated >

- TESTEXCEPTVECT
- TESTINTVECT (for RX700 and RX600 series only)

For users of RX71M, RX72M, RX72N, RX64M, RX65N, RX651, RX66N, RX660, RX671: Before executing the R_CPU_Diag100 function, please call the R_CPU_Diag_IntControl function just for once. The prototype of this function is defined as follows:

void R_CPU_Diag_IntControl(int32_t result, int32_t is_wp);*

Table 3.11 R_CPU_Diag_IntControl Function Interface

Type	C Type	Parameter	Description
input	int32_t*	result	0 : Succeeded 1 : Failed
input	int32_t	is_wp	0 : Write nothing to SLIPRCR register. 1 : Write 1 to SLIPRCR register.

To update software interrupt after the execution of this function, set is_wp to 0. No update is performed when is_wp is 1.

3.3.1.3 Interrupt Disabled Section

For any R_CPU_Diag functions, no interrupt is allowed from beginning to the end of its execution. If you would like to call Test API Function during the execution, please disable interrupts beforehand. Also, set the interrupt level to 7 or less for the interrupt request of the built-in peripheral module that may occur while performing the CPU test "R_CPU_Diag28".

3.3.2 Usage conditions

Table 3.12 summarizes usage conditions.

Table 3.12 Conditions of Use

ID	Topic	Condition	Description
1	Interrupt	Correct execution of the SW	Before calling Test API Functions, user should disable interrupts.
2	ROM cache	Correct execution of the SW	Disable the ROM cache before executing Diagnostic Software.
3	CPU mode	Correct execution of the SW	Execute Diagnostic Software in supervisor mode.
4	Stack	Correct execution of the SW	Use Interrupt Stack Pointer as stack pointer for the function call.
5	Interrupt	Correct execution of the SW	Do not use #175 and #176 of Software Configurable Interrupt B in user code; these are used by interrupt test. NOTE: Not applied to MCU group not supporting the software interrupt 2 activation register (SWINT2R).
6	Environment	Avoid corruption of SW flow by corrupting a control flow variable.	Do not write to the testInt / testExcp variable other than the test procedures specified in Section 3.3.1.2.

3.4 Directives for Software Configuration

Table 3.13 and Table 3.14 show directives to define RX MCU type and for what endian type the RX MCU is configured. The directives can be found in the testCPU_config.h and testCPU_config.inc.

Table 3.13 Directives for Software Configuration (1 of 2)

File name	Directive	Description
testCPU_config.h	RX_DEVICE_TYPE	Specifies the type of RX MCU. Definitions : RX71M : DEVICE_RX71M RX72M : DEVICE_RX72M RX72N : DEVICE_RX72N RX72T : DEVICE_RX72T RX64M : DEVICE_RX64M RX65N : DEVICE_RX65N_651 RX651 : DEVICE_RX65N_651 RX66T : DEVICE_RX66T RX66N : DEVICE_RX66N RX660 : DEVICE_RX660 RX671 : DEVICE_RX671 RX230 : DEVICE_RX231_230 RX231 : DEVICE_RX231_230 RX23E-A : DEVICE_RX23E_A RX23T : DEVICE_RX23T RX23W : DEVICE_RX23W RX24T : DEVICE_RX24T_24U RX24U : DEVICE_RX24T_24U [Example] When RX72M is used : #define RX_DEVICE_TYPE (DEVICE_RX72M)
	ENDIANESS_TYPE_BIG ENDIANESS_TYPE_LITTLE	For little endian enable the ENDIANESS_TYPE_LITTLE disable the ENDIANESS_TYPE_BIG For big endian enable the ENDIANESS_TYPE_BIG disable the ENDIANESS_TYPE_LITTLE [Example] When RX MCU is used in little endian: #define ENDIANESS_TYPE_LITTLE NOTE: Do not define the ENDIANNESS_TYPE_BIG.

Table 3.14 Directives for Software Configuration (2 of 2)

File name	Directive	Description
testCPU_config.inc	RX_DEVICE_TYPE	Specifies the type of RX MCU. Definitions : RX71M : DEVICE_RX71M RX72M : DEVICE_RX72M RX72N : DEVICE_RX72N RX72T : DEVICE_RX72T RX64M : DEVICE_RX64M RX65N : DEVICE_RX65N_651 RX651 : DEVICE_RX65N_651 RX66T : DEVICE_RX66T RX66N : DEVICE_RX66N RX660 : DEVICE_RX660 RX671 : DEVICE_RX671 RX230 : DEVICE_RX231_230 RX231 : DEVICE_RX231_230 RX23E-A : DEVICE_RX23E_A RX23T : DEVICE_RX23T RX23W : DEVICE_RX23W RX24T : DEVICE_RX24T_24U RX24U : DEVICE_RX24T_24U [Example] When RX72M is used : RX_DEVICE_TYPE .EQU DEVICE_RX72M
	ENDIANESS_TYPE_BIG_ASM ENDIANESS_TYPE_LITTLE_ASM	For little endian Set 1 to the ENDIANESS_TYPE_LITTLE_ASM Set 0 to the ENDIANESS_TYPE_BIG_ASM For big endian Set 0 to the ENDIANESS_TYPE_BIG_ASM Set 1 to the ENDIANESS_TYPE_LITTLE_ASM [Example] When RX MCU is used in little endian: ENDIANESS_TYPE_BIG_ASM .EQU 0 ENDIANESS_TYPE_LITTLE_ASM .EQU 1

NOTE: A warning message may appear when compiling the software due to alignment mismatch in section P. In such case, however, you can ignore the message because compiler will handle it correctly.

3.5 Software Package

Table 3.15 shows description of each design file.

Table 3.15 Design Files

#	File name	Description
1	testCPU.h	Declaration of Diagnostic SW API (R_CPU_Diag function to be called by the application SW)
2	r_cpu_diag_rx.c	Definition of API functions and vector table used for diagnostics.
3	testCPU_config.h	Compile option definitions, through which it is possible to specify MCU device and endianness.
4	testCPU_config.inc	Compile option definitions, through which it is possible to specify MCU device and endianness.
5	r_cpu_diag_rx_private.h	Definition of the LUT
6	globVarAsm.inc	Definition values used for assembler.
7	intCtrl.h	Declaration of the intCtrl().
8	intCtrl.src	Definition of the intCtrl().
9	romCacheCtrl	Definition of the romCacheCtrl().
10	romCacheCtrl.src	Declaration of the romCacheCtrl().
11	_signatureCrt.src	Definition of the sirgantureCrt().
12	r_cpu_diag_NNN_rx.h	Declaration of each R_CPU_Diag function. NNN = 0-104.
13	r_cpu_diag_NNN_rx.src	Definition of each R_CPU_Diag function. NNN = 0-104.

3.6 Resource Usage

Table 3.16 provides an overview of the RX72M memory resources used by little endian.

Note that resources related to the main function are not included.

Maximum stack usage is 256 bytes.

Also note that dynamic memory allocation is not implemented.

Table 3.16 Memory Resource (RX72M)

Object	ROM		RAM
	Code	Data	
r_cpu_diag_NNN_rx.obj, r_cpu_diag_rx.obj, _signatureCrt.obj, romCacheCtrl.obj, intCtrl.obj (NNN = 0-104)	41.2 KBytes	13.3 KBytes	8 bytes

Table 3.17 - Table 3.25 describe the execution time for each R_CPU_Diag function. In this table, execution time of R_CPU_Diag functions of "LUT" type is noted as a whole data set.

Table 3.17 Execution Time (Calculation Cycle Count) (1 of 9)

Test#	Data Set #	RX71M	RX72T	RX72M RX72N	RX64M	RX65N RX651	RX66T	RX66N	RX660	RX671	RX230 RX231 RX23T RX23W	RX23E-A	RX24T RX24U
0		236	216	216	217	275	236	216	201	216	255	215	295
1		330	310	310	308	409	330	310	293	307	361	307	442
2		1,168	1,170	1,170	1,150	1,270	1,168	1,170	1,157	1,171	1,199	1,149	1,393
3		268	246	238	240	310	260	238	220	234	277	238	337
4		730	742	742	697	873	730	742	706	740	767	695	967
5		296	274	268	271	352	288	268	247	264	309	269	376
6		416	400	396	367	538	414	396	347	375	467	365	583
7		308	314	314	295	336	308	314	303	313	319	294	364
8		224	210	208	206	277	226	208	190	206	245	205	298
9		336	320	324	317	382	340	324	298	311	347	314	382
10		214	218	216	196	250	214	216	203	216	225	195	271
11		488	468	472	475	551	488	472	459	471	497	473	556
12		302	306	306	285	359	302	306	292	307	315	284	373
13		274	256	254	258	334	274	254	243	252	281	256	334
14		272	254	252	254	330	272	252	239	252	281	253	340
15		170	154	154	156	196	170	154	141	153	181	155	211

Table 3.18 Execution Time (Calculation Cycle Count) (2 of 9)

Test#	Data Set #	RX71M	RX72T	RX72M RX72N	RX64M	RX65N RX651	RX66T	RX66N	RX660	RX671	RX230 RX231 RX23T RX23W	RX23E-A	RX24T RX24U
16		266	270	270	246	303	268	270	253	266	275	244	325
17		946	924	926	930	1,013	944	926	913	923	947	927	1,009
18		1,152	1,132	1,132	1,137	1,280	1,152	1,132	1,121	1,139	1,163	1,135	1,291
19		1,024	1,028	1,028	1,009	1,147	1,026	1,028	1,016	1,032	1,035	1,007	1,126
20		1,198	1,182	1,180	1,143	1,366	1,200	1,180	1,126	1,168	1,187	1,139	1,345
21		1,198	1,180	1,180	1,136	1,401	1,200	1,180	1,119	1,185	1,195	1,134	1,330
22		944	928	928	919	1,050	948	928	903	926	955	917	1,015
23		738	716	716	724	778	738	716	707	724	745	722	766
24		730	716	716	716	792	736	716	706	720	737	714	802
25		854	808	808	797	1,116	826	808	780	810	863	795	1,204
26		366	352	352	338	452	370	352	323	343	381	337	475
27		280	286	286	265	293	284	286	271	284	287	263	310
28		284	286	288	269	350	284	288	275	285	287	266	343
29		640	668	668	532	875	666	668	538	591	689	529	907
30		764	744	744	728	843	764	744	711	726	759	726	844
31		734	712	712	720	783	734	712	703	714	743	718	838
32		972	956	956	872	1,194	976	956	855	942	975	870	1,177
33		-	630	632	-	-	-	632	602	612	-	-	-
34		-	644	646	-	-	-	646	617	627	-	-	-
35		-	-	764	-	-	-	764	733	744	-	-	-
36		-	-	254	-	-	-	254	-	249	-	-	-
37		-	-	260	-	-	-	260	-	254	-	-	-
38		-	-	90	-	-	-	90	-	96	-	-	-
39		-	-	112	-	-	-	112	-	114	-	-	-
40		-	-	302	-	-	-	302	-	323	-	-	-
41		-	762	162	-	-	-	162	-	185	-	-	-
50	0	852	884	958	726	1,148	918	958	660	702	1,021	716	1,279
50	1	884	884	958	726	1,148	918	958	661	701	1,021	716	1,279
50	2	852	884	958	726	1,148	918	958	661	703	1,021	716	1,279
50	3	884	884	958	726	1,148	918	958	661	700	1,021	716	1,279
50	4	852	884	958	726	1,148	918	958	661	702	1,021	716	1,279

Table 3.19 Execution Time (Calculation Cycle Count) (3 of 9)

Test#	Data Set #	RX71M	RX72T	RX72M RX72N	RX64M	RX65N RX651	RX66T	RX66N	RX660	RX671	RX230 RX231 RX23T RX23W	RX23E-A	RX24T RX24U
50	5	884	884	958	726	1,148	918	958	661	701	1,021	716	1,279
51	0	806	882	966	687	1,121	900	966	670	699	1,021	685	1,252
51	1	846	882	966	687	1,121	900	966	660	702	1,021	685	1,252
51	2	806	882	966	687	1,121	900	966	670	702	1,021	685	1,252
51	3	846	882	966	687	1,121	900	966	660	702	1,021	685	1,252
51	4	806	882	966	687	1,121	900	966	670	699	1,021	685	1,252
51	5	846	882	966	687	1,121	900	966	660	702	1,021	685	1,252
52	0	866	882	966	705	1,038	884	966	661	706	1,019	703	1,282
52	1	886	882	966	705	1,038	884	966	661	709	1,019	703	1,282
52	2	866	882	966	705	1,038	884	966	661	709	1,019	703	1,282
52	3	886	882	966	705	1,038	884	966	661	708	1,019	703	1,282
52	4	866	882	966	705	1,038	884	966	661	706	1,019	703	1,282
52	5	886	882	966	705	1,038	884	966	661	709	1,019	703	1,282
53	0	814	880	956	696	1,121	900	956	662	707	1,019	694	1,252
53	1	882	880	956	696	1,121	900	956	662	704	1,019	694	1,252
53	2	814	880	956	696	1,121	900	956	662	706	1,019	694	1,252
53	3	882	880	956	696	1,121	900	956	662	704	1,019	694	1,252
53	4	814	880	956	696	1,121	900	956	662	707	1,019	694	1,252
53	5	882	880	956	696	1,121	900	956	662	704	1,019	694	1,252
54	0	886	870	1,044	756	1,222	888	1,044	732	770	1,099	754	1,282
54	1	926	870	1,044	756	1,222	888	1,044	732	772	1,099	754	1,282
54	2	886	870	1,044	756	1,222	888	1,044	732	772	1,099	754	1,282
54	3	926	870	1,044	756	1,222	888	1,044	732	772	1,099	754	1,282
54	4	886	870	1,044	756	1,222	888	1,044	732	770	1,099	754	1,282
54	5	926	870	1,044	756	1,222	888	1,044	732	772	1,099	754	1,282
55	0	892	910	1,028	758	1,219	888	1,028	740	772	1,101	756	1,282
55	1	918	910	1,028	758	1,219	888	1,028	730	772	1,101	756	1,282
55	2	892	910	1,028	758	1,219	888	1,028	740	772	1,101	756	1,282
55	3	918	910	1,028	758	1,219	888	1,028	730	771	1,101	756	1,282
55	4	892	910	1,028	758	1,219	888	1,028	740	772	1,101	756	1,282
55	5	918	910	1,028	758	1,219	888	1,028	730	772	1,101	756	1,282

Table 3.20 Execution Time (Calculation Cycle Count) (4 of 9)

Test#	Data Set #	RX71M	RX72T	RX72M RX72N	RX64M	RX65N RX651	RX66T	RX66N	RX660	RX671	RX230 RX231 RX23T RX23W	RX23E-A	RX24T RX24U
56	0	814	842	930	665	1,059	840	930	616	654	961	647	1,189
56	1	816	842	930	665	1,059	840	930	611	655	961	647	1,189
56	2	814	842	930	665	1,059	840	930	616	657	961	647	1,189
56	3	816	842	930	665	1,059	840	930	611	655	961	647	1,189
56	4	814	842	930	665	1,059	840	930	616	654	961	647	1,189
56	5	816	842	930	665	1,059	840	930	611	655	961	647	1,189
57	0	1,076	1,026	1,184	942	1,223	1,044	1,184	886	929	1,233	940	1,420
57	1	1,052	1,004	1,162	920	1,201	1,022	1,162	864	904	1,215	918	1,402
57	2	1,158	1,108	1,266	1,024	1,305	1,126	1,266	968	1,009	1,317	1,022	1,513
57	3	1,310	1,262	1,420	1,178	1,459	1,280	1,420	1,122	1,161	1,469	1,176	1,678
57	4	594	560	642	520	676	578	642	484	513	677	518	793
58	0	982	930	1,086	846	1,160	948	1,086	789	832	1,147	844	1,354
58	1	972	924	1,080	840	1,127	942	1,080	783	824	1,129	838	1,321
58	2	1,082	1,030	1,186	946	1,296	1,048	1,186	889	932	1,249	944	1,441
58	3	1,210	1,162	1,318	1,078	1,356	1,180	1,318	1,021	1,062	1,369	1,076	1,570
58	4	542	506	586	466	625	522	586	429	460	621	464	733
59	0	654	620	662	567	830	638	662	544	564	711	564	841
59	1	630	620	662	567	830	638	662	544	561	711	564	841
59	2	654	620	662	567	830	638	662	544	564	711	564	841
59	3	630	620	662	567	830	638	662	544	562	711	564	841
59	4	654	620	662	567	830	638	662	544	564	711	564	841
59	5	630	620	662	567	830	638	662	544	561	711	564	841
59	6	654	620	662	567	830	638	662	544	564	711	564	841
59	7	630	620	662	567	830	638	662	544	562	711	564	841
59	8	654	620	662	567	830	638	662	544	564	711	564	841
59	9	630	620	662	567	830	638	662	544	561	711	564	841
59	10	654	620	662	567	830	638	662	544	564	711	564	841
59	11	630	620	662	567	830	638	662	544	562	711	564	841
59	12	654	620	662	567	830	638	662	544	564	711	564	841
59	13	528	514	550	473	691	532	550	451	467	593	470	700
59	14	546	514	550	473	691	532	550	451	469	593	470	700

Table 3.21 Execution Time (Calculation Cycle Count) (5 of 9)

Test#	Data Set #	RX71M	RX72T	RX72M RX72N	RX64M	RX65N RX651	RX66T	RX66N	RX660	RX671	RX230 RX231 RX23T RX23W	RX23E-A	RX24T RX24U
59	15	528	514	550	473	691	532	550	451	468	593	470	700
60	0	642	618	660	565	821	638	660	545	569	709	562	829
60	1	626	618	660	565	821	638	660	545	565	709	562	829
60	2	642	618	660	565	821	638	660	545	568	709	562	829
60	3	626	618	660	565	821	638	660	545	566	709	562	829
60	4	642	618	660	565	821	638	660	545	569	709	562	829
60	5	626	618	660	565	821	638	660	545	565	709	562	829
60	6	642	618	660	565	821	638	660	545	568	709	562	829
60	7	626	618	660	565	821	638	660	545	566	709	562	829
60	8	642	618	660	565	821	638	660	545	569	709	562	829
60	9	626	618	660	565	821	638	660	545	565	709	562	829
60	10	642	618	660	565	821	638	660	545	568	709	562	829
60	11	626	618	660	565	821	638	660	545	566	709	562	829
60	12	642	618	660	565	821	638	660	545	569	709	562	829
60	13	524	512	548	471	683	532	548	452	471	591	468	691
60	14	536	512	548	471	683	532	548	452	475	591	468	691
60	15	524	512	548	471	683	532	548	452	472	591	468	691
61	0	642	618	660	565	821	638	660	545	567	709	562	829
61	1	622	618	660	565	821	638	660	545	566	709	562	829
61	2	642	618	660	565	821	638	660	545	568	709	562	829
61	3	622	618	660	565	821	638	660	545	566	709	562	829
61	4	642	618	660	565	821	638	660	545	567	709	562	829
61	5	622	618	660	565	821	638	660	545	566	709	562	829
61	6	642	618	660	565	821	638	660	545	568	709	562	829
61	7	622	618	660	565	821	638	660	545	566	709	562	829
61	8	642	618	660	565	821	638	660	545	567	709	562	829
61	9	622	618	660	565	821	638	660	545	566	709	562	829
61	10	642	618	660	565	821	638	660	545	568	709	562	829
61	11	622	618	660	565	821	638	660	545	566	709	562	829
61	12	642	618	660	565	821	638	660	545	567	709	562	829
61	13	522	512	548	471	683	532	548	452	471	591	468	691

Table 3.22 Execution Time (Calculation Cycle Count) (6 of 9)

Test#	Data Set #	RX71M	RX72T	RX72M RX72N	RX64M	RX65N RX651	RX66T	RX66N	RX660	RX671	RX230 RX231 RX23T RX23W	RX23E-A	RX24T RX24U
61	14	538	512	548	471	683	532	548	452	473	591	468	691
61	15	522	512	548	471	683	532	548	452	473	591	468	691
62	0	984	958	1,010	915	1,129	976	1,010	895	918	1,059	912	1,159
62	1	974	958	1,010	915	1,129	976	1,010	895	915	1,059	912	1,159
62	2	984	958	1,010	915	1,129	976	1,010	895	919	1,059	912	1,159
62	3	974	958	1,010	915	1,129	976	1,010	895	918	1,059	912	1,159
62	4	984	958	1,010	915	1,129	976	1,010	895	918	1,059	912	1,159
62	5	974	958	1,010	915	1,129	976	1,010	895	915	1,059	912	1,159
62	6	984	958	1,010	915	1,129	976	1,010	895	919	1,059	912	1,159
62	7	974	958	1,010	915	1,129	976	1,010	895	918	1,059	912	1,159
62	8	984	958	1,010	915	1,129	976	1,010	895	918	1,059	912	1,159
62	9	974	958	1,010	915	1,129	976	1,010	895	915	1,059	912	1,159
62	10	984	958	1,010	915	1,129	976	1,010	895	919	1,059	912	1,159
62	11	974	958	1,010	915	1,129	976	1,010	895	918	1,059	912	1,159
62	12	984	958	1,010	915	1,129	976	1,010	895	918	1,059	912	1,159
62	13	802	786	828	751	928	802	828	732	753	871	748	955
62	14	810	786	828	751	928	802	828	732	754	871	748	955
62	15	802	786	828	751	928	802	828	732	752	871	748	955
63	0	1,136	1,166	1,330	1,006	1,402	1,184	1,330	992	1,022	1,381	1,005	1,585
64	0	1,102	1,090	1,248	949	1,397	1,106	1,248	912	955	1,285	947	1,531
65	0	944	922	958	905	1,072	944	958	886	904	989	902	1,054
65	1	944	922	958	905	1,072	944	958	886	904	989	902	1,054
65	2	944	922	958	905	1,072	944	958	886	904	989	902	1,054
65	3	944	922	958	905	1,072	944	958	886	904	989	902	1,054
65	4	944	922	958	905	1,072	944	958	886	904	989	902	1,054
65	5	944	922	958	905	1,072	944	958	886	904	989	902	1,054
65	6	944	922	958	905	1,072	944	958	886	904	989	902	1,054
65	7	944	922	958	905	1,072	944	958	886	904	989	902	1,054
65	8	944	922	958	905	1,072	944	958	886	904	989	902	1,054
65	9	944	922	958	905	1,072	944	958	886	904	989	902	1,054
66	0	1,036	1,012	1,106	973	1,250	1,054	1,106	933	961	1,173	970	1,267

Table 3.23 Execution Time (Calculation Cycle Count) (7 of 9)

Test#	Data Set #	RX71M	RX72T	RX72M RX72N	RX64M	RX65N RX651	RX66T	RX66N	RX660	RX671	RX230 RX231 RX23T RX23W	RX23E-A	RX24T RX24U
66	1	1,056	1,012	1,106	973	1,250	1,054	1,106	933	960	1,173	970	1,267
67	0	1,108	1,072	1,080	1,090	1,182	1,094	1,080	1,057	1,073	1,119	1,087	1,165
67	1	1,110	1,072	1,080	1,090	1,182	1,094	1,080	1,057	1,074	1,119	1,087	1,165
67	2	1,108	1,072	1,080	1,090	1,182	1,094	1,080	1,057	1,077	1,119	1,087	1,165
67	3	1,110	1,072	1,080	1,090	1,182	1,094	1,080	1,057	1,076	1,119	1,087	1,165
67	4	1,108	1,072	1,080	1,090	1,182	1,094	1,080	1,057	1,073	1,119	1,087	1,165
67	5	1,110	1,072	1,080	1,090	1,182	1,094	1,080	1,057	1,074	1,119	1,087	1,165
67	6	1,108	1,072	1,080	1,090	1,182	1,094	1,080	1,057	1,077	1,119	1,087	1,165
67	7	1,048	1,012	1,018	1,028	1,117	1,032	1,018	996	1,015	1,057	1,025	1,102
68	0	1,064	994	1,006	1,012	1,153	1,014	1,006	978	994	1,043	1,009	1,117
68	1	1,064	994	1,006	1,012	1,153	1,014	1,006	978	994	1,043	1,009	1,117
68	2	1,064	994	1,006	1,012	1,153	1,014	1,006	978	994	1,043	1,009	1,117
68	3	1,064	994	1,006	1,012	1,153	1,014	1,006	978	994	1,043	1,009	1,117
68	4	1,064	994	1,006	1,012	1,153	1,014	1,006	978	994	1,043	1,009	1,117
68	5	1,122	1,050	1,062	1,069	1,216	1,070	1,062	1,034	1,050	1,101	1,066	1,177
68	6	1,122	1,050	1,062	1,069	1,216	1,070	1,062	1,034	1,050	1,101	1,066	1,177
68	7	1,122	1,050	1,062	1,069	1,216	1,070	1,062	1,034	1,050	1,101	1,066	1,177
68	8	1,122	1,050	1,062	1,069	1,216	1,070	1,062	1,034	1,050	1,101	1,066	1,177
68	9	1,122	1,050	1,062	1,069	1,216	1,070	1,062	1,034	1,050	1,101	1,066	1,177
68	10	1,122	1,050	1,062	1,069	1,216	1,070	1,062	1,034	1,050	1,101	1,066	1,177
68	11	1,122	1,050	1,062	1,069	1,216	1,070	1,062	1,034	1,050	1,101	1,066	1,177
68	12	1,122	1,050	1,062	1,069	1,216	1,070	1,062	1,034	1,050	1,101	1,066	1,177
68	13	1,122	1,050	1,062	1,069	1,216	1,070	1,062	1,034	1,050	1,101	1,066	1,177
69	0	1,080	1,018	1,040	1,035	1,156	1,052	1,040	1,003	1,017	1,067	1,032	1,096
69	1	1,080	1,018	1,040	1,035	1,156	1,052	1,040	1,003	1,017	1,067	1,032	1,096
69	2	1,080	1,018	1,040	1,035	1,156	1,052	1,040	1,003	1,017	1,067	1,032	1,096
69	3	1,080	1,018	1,040	1,035	1,156	1,052	1,040	1,003	1,017	1,067	1,032	1,096
69	4	1,080	1,018	1,040	1,035	1,156	1,052	1,040	1,003	1,017	1,067	1,032	1,096
69	5	1,080	1,018	1,040	1,035	1,156	1,052	1,040	1,003	1,017	1,067	1,032	1,096
69	6	1,080	1,018	1,040	1,035	1,156	1,052	1,040	1,003	1,017	1,067	1,032	1,096
69	7	1,006	948	968	963	1,078	980	968	932	946	995	960	1,024
69	8	1,006	948	968	963	1,078	980	968	932	946	995	960	1,024

Table 3.24 Execution Time (Calculation Cycle Count) (8 of 9)

Test#	Data Set #	RX71M	RX72T	RX72M RX72N	RX64M	RX65N RX651	RX66T	RX66N	RX660	RX671	RX230 RX231 RX23T RX23W	RX23E-A	RX24T RX24U
69	9	932	876	896	891	1,000	908	896	861	875	923	888	952
70	0	912	868	904	894	1,063	886	904	853	870	925	892	1,126
70	1	912	868	904	894	1,063	886	904	853	870	925	892	1,126
70	2	912	868	904	894	1,063	886	904	853	870	925	892	1,126
70	3	912	868	904	894	1,063	886	904	853	870	925	892	1,126
71	0	946	910	934	927	1,025	930	934	896	911	959	925	1,018
71	1	930	894	920	912	1,010	914	920	881	896	945	910	1,018
71	2	926	890	914	907	1,005	910	914	876	891	939	905	1,009
71	3	926	890	916	908	1,006	910	916	877	892	941	906	1,015
71	4	890	854	878	871	966	874	878	841	856	903	869	964
71	5	896	860	884	877	972	880	884	847	862	909	875	964
72	0	-	212	212	-	-	-	212	-	204	-	-	-
72	1	-	212	212	-	-	-	212	-	204	-	-	-
72	2	-	212	212	-	-	-	212	-	204	-	-	-
72	3	-	212	212	-	-	-	212	-	205	-	-	-
72	4	-	212	212	-	-	-	212	-	204	-	-	-
72	5	-	212	212	-	-	-	212	-	204	-	-	-
72	6	-	212	212	-	-	-	212	-	204	-	-	-
72	7	-	212	212	-	-	-	212	-	205	-	-	-
72	8	-	212	212	-	-	-	212	-	204	-	-	-
72	9	-	212	212	-	-	-	212	-	204	-	-	-
72	10	-	212	212	-	-	-	212	-	204	-	-	-
72	11	-	212	212	-	-	-	212	-	205	-	-	-
72	12	-	212	212	-	-	-	212	-	204	-	-	-
72	13	-	212	212	-	-	-	212	-	204	-	-	-
72	14	-	212	212	-	-	-	212	-	204	-	-	-
73	0	-	222	224	-	-	-	224	-	213	-	-	-
73	1	-	222	224	-	-	-	224	-	213	-	-	-
73	2	-	222	224	-	-	-	224	-	214	-	-	-
73	3	-	222	224	-	-	-	224	-	214	-	-	-
73	4	-	222	224	-	-	-	224	-	213	-	-	-
73	5	-	222	224	-	-	-	224	-	213	-	-	-

Table 3.25 Execution Time (Calculation Cycle Count) (9 of 9)

Test#	Data Set #	RX71M	RX72T	RX72M RX72N	RX64M	RX65N RX651	RX66T	RX66N	RX660	RX671	RX230 RX231 RX23T RX23W	RX23E-A	RX24T RX24U
73	6	-	222	224	-	-	-	224	-	214	-	-	-
73	7	-	222	224	-	-	-	224	-	214	-	-	-
73	8	-	222	224	-	-	-	224	-	213	-	-	-
73	9	-	222	224	-	-	-	224	-	213	-	-	-
73	10	-	222	224	-	-	-	224	-	214	-	-	-
73	11	-	222	224	-	-	-	224	-	214	-	-	-
73	12	-	222	224	-	-	-	224	-	213	-	-	-
73	13	-	222	224	-	-	-	224	-	213	-	-	-
73	14	-	222	224	-	-	-	224	-	214	-	-	-
100		722	674	704	610	659	694	704	589	601	-	-	-
101		1,074	1,064	1,076	1,024	1,188	1,082	1,076	1,001	1,019	1,127	1,021	1,231
102		354	338	342	331	394	358	342	312	325	377	328	421
103		850	836	772	820	925	856	772	796	738	901	817	967
104		814	810	834	778	904	828	834	755	777	873	774	916

3.7 Requirements for Safety Relevant Applications

Table 3.26 shows requirements of use in safety relevant applications.

Table 3.26 Requirements for Safety Relevant Applications

ID	Topic	Sub-topic	Description
SW_1	SW integration	Function call and return	On the return of API function, user shall check the variable "result" to see whether the test is passed or failed, as well as compare the returned signature value stored in the signatureVector with the expected signature value.
SW_2	SW integration	Function call	When calling the test API function more than once, prepare variables to store the returned output (i.e. result and signatureVector) as many as number of function calls. If the same variables are used for multiple times, variable shall be initialized to zero each time before the next function call.
SW_3	SW integration	Function environment	Before calling the test API function, variable to store the result value shall be initialize to zero.
SW_4	SW integration	Interrupt	Before calling the test API function, variable to store the result value shall be initialize to zero.
PR_1	Project management	User expertise	User must have good expertise on embedded programming on the target MCU SW, assembly programming and C.

3.8 Diagnostic Coverage and Watchdog

Diagnostic coverage is calculated from all CPU tests, by executing all R_CPU_Diag functions of type Fixed/LUT/Specific with valid dataSet value (see Table 3.7).
Diagnostic coverage also considers errors detected by Watchdog. Watchdog therefore is necessary for CPU Core Test.
Table 3.27 summarizes the recommendation for implementing Watchdog.
Some hardware faults may cause damage to control flow. In such case, the fault can be detected by Watchdog instead.
CPU Core Test also has control flows that trigger activation of such faults, however, as stated above, implementation of Watchdog is essential to make fault detection accurate and complete.

Table 3.27 Recommendation of Watchdog Implementation

ID	Topic	Description	Remarks
1	WD refresh	Consider a control flow monitoring for the WD refresh function: the refresh is done only if the control flow mechanism (e.g. proper value of global variable) is respected.	
2	WD refresh	Consider a strategy as the following: activate the WD refresh only if all the main tasks having a predictable and periodic timing schedule of the application SW are called in the proper order.	

4. Internal RAM Test

4.1 Test Objectives

The objective of RAM Test is to verify the internal RAM of the MCU.

Main features of CPU Test are described as below:

a. Whole memory check including stack(s)
   - Memory size programmable at compile time.

b. Block-wise implementation of the test
   - Size of block programmable at compile time.

c. Support of two testing algorithms
   - Extended March C -
   - WALPAT

d. Supports destructive / non-destructive memory testing.

See Appendix B of Section 6.2 for details about test algorithms.

4.2 Test strategy

In the RAM Test, the internal RAM, extended RAM, and ECC RAM (data only, ECC code is excluded) are tested by test block units. There are three test areas: Area 1 for internal RAM, area 2 for extended RAM, and area 3 for ECC RAM.

Please refer to the reference documents [7] - [22] to see if the MCU has extended RAM and/or ECC RAM.

Memory size and block size can be specified according to the use of device and application.

- MUTSize (MUTSizeX (X=1,2,3))
  - Size of memory under test expressed in 4-byte length units.
- BUTSize (BUTSizeX (X=1,2,3))
  - Size of block under test expressed in 4-byte length units.
- numberOfBUT (numberOfBUT X (X=1,2,3))
  - Number of blocks to which the memory is divided.
- startAddress (startAddressX (X=1,2,3))
  - Start address to the memory under test.

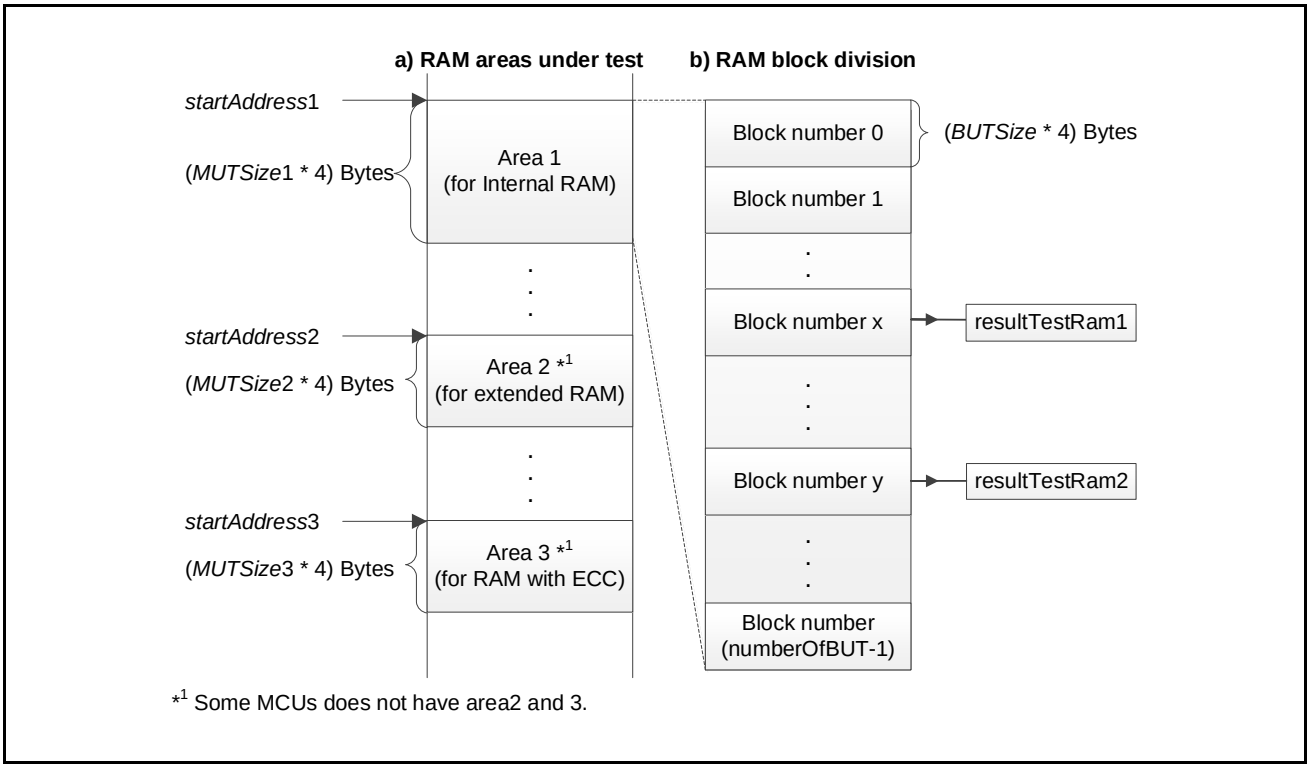

Figure 4.1 Memory Area

Figure 4.1 shows how each memory area divided into a given number of blocks (numberOfBUT).
Each test block is identified with index ranging from 0 to (numberOfBUT - 1).
Each block can be tested in a destructive or non-destructive method.
When using non-destructive method, one RAM block shall be reserved as a buffer to temporarily save the content of the block to be tested.
The buffer itself can also be tested. In that case, buffer shall be tested before the any other blocks and by destructive method.
User also need to reserve the memory area to store the test result. See 4.4.1.2 for details of how to reserve buffer area.

The result of the test is stored in two variables on the unused RAM locations accessible from the application software (see Section 4.4). The two variables to store test result is placed at fixed addresses and shall be placed in different blocks.
This strategy is to avoid the issue in which faults cannot be detected because the results are stored in the same faulty block.
By placing variables in different blocks, the either block is still available even when a fault is occurred to the other one. See 4.4.1.3 for how to reserve variables to store test result (resultTestRam1, resultTestRam2).
User shall check out the values of the result variables after the test is complete.
Judgement is performed as shown below:

- resultTestRam1 = resultTestRam2 = 1 means test is passed.
- any other combination means the test failed.

See section 4.4 for how to check test result and how to reserve test result variables.

4.3 API and RAM Test Environment

RAM block test is called through a testRAM function. The testRAM is defined as follows:

void testRAM(uint32_t area, uint32_t rambuffer, uint32_t index, uint32_t selectAlgorithm, uint32_t destructive)

Table 4.1 shows details of the testRAM function interface.

Table 4.1 testRAM Function Interface

#	Type	C Type	Parameter	Description
1	input	uint32_t	Area	RAM area under test "0" : Internal RAM "1" : Extended RAM "2" : ECC RAM Others : Parameter error
2	input	uint32_t	rambuffer	Whether RAM write buffer is used or not "0" : Used "1" : Not used Others : Not used See reference documents [7] - [22] for which RAM area supports RAM write buffer function.
3	Input	uint32_t	index	RAM block under test (0 - numberOfBUT-1)
4	input	uint32_t	selectAlgorithm	Test algorithm "0" : Extended March C- "1" : WALPAT Others : WALPAT
5	input	uint32_t	destructive	Test method "0" : Non-destructive (Saves RAM block value to buffer during test) "1" : Destructive (RAM block is initialized to all 0 after the test) Others : Non-destructive

RAM Test is performed on RAM area specified by index parameter using algorithm specified by selectAlgorithm, by RAM block unit specified by BUTSize.
Valid value of index parameter is 0 - numberOfBUT-1.
numberOfBUT indicates the number of blocks derived by dividing the memory size by block size specified by BUTSize.
If invalid value (greater than (numberOfBUT - 1)) is specified in the index parameter, 0 (FAIL) is returned.

4.4 Software Integration Rules

This section provides guidelines for how to integrate the RAM Test within the user project.

4.4.1 Code Integration

4.4.1.1 Definition of Memory Size and Block Size

The user shall set the size of the RAM under test, start address and block size for each RAM area under test.
Define this information from directives in the testRAM_config.h.
Table 4.2 shows relationship between BUTSize and number of blocks.

Table 4.2 Relationship between BUTSize and Number of Blocks

BUT Size	Number of blocks	Index
MUTSize/4	4	0, 1, 2, 3
MUTSize/8	8	0, 1, 2, 3, 4, 5, 6, 7
MUTSize/16	16	0, 1, 2, 3, 4, ..., 15
MUTSize/32	32	0, 1, 2, 3, 4, ..., 31
MUTSize/64	64	0, 1, 2, 3, 4, ..., 63
・・・	・・・	・・・
MUTSize/MUTSize	MUTSize	0, 1, 2, 3, 4, ..., MUTSize-1

The following example shows how the RAM area 1 (128 KBytes) is divided by 1-KByte block units.

```
/*-------------------------------------------------------------*/
/* RAM test area definition (internal RAM)                     */
/*-------------------------------------------------------------*/
/*size of the RAM Memory Under Test: 128 KByte = 128 * 1024 bytes = 131072 bytes    = 32768 double words */
#define MUTSize1                    (32768U)
/* size of the Block of RAM Under Test of 1 KByte */
#define BUTSize1                    (MUTSize1/128U)
/* number of block in which the RAM memory is divided */
#define numberOfBUT1                (MUTSize1/BUTSize1)
/* start address of the RAM Memory under test area */
#define startAddress1               (0x00000000U)
```

4.4.1.2 Reserving Buffer

A buffer can be placed at the top of the RAM area. Specify the start address and the size of the buffer by testRAM_config.h. When the start address of the buffer is 0x00000000 and buffer size is 1 KByte (1024 bytes), definition is as shown below.

```
/*----------------------------------------------------------------*/
/* Address definition of RAM test temporary data area             */
/*----------------------------------------------------------------*/
#define addressBufferSize               (1024U)
#pragma address addressBuffer = 0x00000000
```

Address of first section of RAM can be figured out by summing the RAM area start address and buffer size. RAM area start address varies depending on MCU. RAM area of this MCU starts from 0x00000000. In this case, the first section of RAM is 0x00000000 + 1 KByte (0x400 bytes) = 0x00000400. Figure 4.2 shows how to specify the start address of the first section of the RAM.

Access to Section management setting of the Linker as below.

Project > C/C++ Project Settings > Tool Settings > Linker > Section

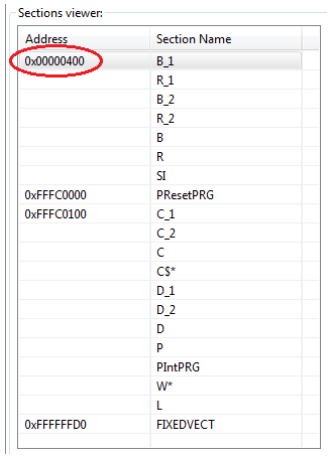

Figure 4.2 Section Management Setting

4.4.1.3 Placing Result Variables

The result of the test is stored in two variables (resultTestRam1 and resultTestRam2) on the unused RAM locations accessible from the application software.
These variables shall be placed at the fixed address.
resultTestRam1 and resultTestRam2 are defined in testRAM.c.
Example: When 128-KByte RAM is divided by 1-KByte block units,

- resultTestRam1 is placed in the last 4-byte length unit in the block 63.
- resultTestRam2 is placed in the last 4-byte length unit in the block 127.

Variables are defined in the testRAM_config.h as shown below:

```
/*------------------------------------------------------------*/
/* Address definition of RAM test result storage area         */
/*------------------------------------------------------------*/
#pragma address resultTestRam1 = 0x0000FFFC
#pragma address resultTestRam2 = 0x0001FFFC
```

4.4.1.4 Addressing ECCRAM-Related Registers

This is how to specify the address of ECCRAM-related registers (ECCRAMMODE and ECCRAMPRCR). This setting is not required for the MCU without ECCRAM. Leave the addresses as default. See User's Manual: Hardware of the MCU for register address and setting value.
Variables are defined in the testRAM_config.h as shown below: Setting value is for RX72M.

```
/*------------------------------------------------------------*/
/* Address definition of ECCRAM operation mode control register */
/*------------------------------------------------------------*/
#pragma address ECCRAMMODE = 0x000812C0

/*------------------------------------------------------------*/
/* Address definition of ECCRAM protect register */
/*------------------------------------------------------------*/
#pragma address ECCRAMPRCR = 0x000812C4
```

4.4.1.5 Test Flow and Test Results

It is recommended to perform buffer test before any other blocks. (NOTE: Buffer data is lost after buffer test regardless of defined test method, destructive or non-destructive)

A recommended flow of RAM Test is as below:

1. Test the buffer via testRAM function
2. Test the rest of blocks via testRAM function

The testRAM function can be used as below:

1. Include the testRAM.h
2. Define input parameter of the testRAM function
    a. *area*
    b. *use RAM write buffer measure*
    c. *index*
    d. *select Algorithm*
    e. *destructive*
3. Disable interruption
4. Call the testRAM function
5. Enable interruption
6. Check result variables

Example:

```
#include "testRAM.h"

uint32_t area = 0;
uint32_t isRamBuf = 0;
uint32_t index = 7;
uint32_t selectAlgorithm = 0;
uint32_t destructive = 0;

/* disable interrupt : CLRPSW I */

testRAM(area, isRamBuf, index, selectAlgorithm, destructive);

/* enable interrupt : SETPSW I */

if(!(resultTestRam1&&resultTestRam2)){ /*Fault detection*/
        errorHandler();
}
```

In the example above, diagnostic SW checks the result value to detect fault after the testRAM function returns. The output of testRAM is stored in two variables (resultTestRam1, resultTestRam2). When resultTestRam1 and resultTestRam2 are both equal to 1, no faults have been detected, otherwise fault handling management should start (calling of errorHandler() function in the above example).

4.4.2 Usage conditions

Table 4.3 summarizes usage conditions.

Table 4.3 Condition of Use

ID	Topic	Condition	Description
1	Interrupt	Avoid corruption of function context	RAM test has interrupt disabled section. When the RAM Test is called, interruption shall be disabled by user.
2	CPU mode	Correct execution of the SW	Execute RAM Test in supervisor mode.
3	Stack	Avoid corruption of the stack	Stack area is tested by non-destructive method.
4	Environment	Avoid corruption of variables storing test results	In any application code other than the RAM Test do not overwrite values of resultTestRam1 and resultTestRam2.
5	Environment	Avoid data lost	Note that the buffer data is lost when RAM Test is called.
6	Configuration	Avoid data lost	Do not place the result variables (resultTestRam1 and resultTestRam2) in the same block where the buffer is placed.
7	Configuration	Compliance with SW test strategy	Address of memory under test shall be aligned by 4-byte length unit.
8	Configuration	Compliance with SW test strategy	BUTSize shall be integer number, respecting the following theory. BUTSize=MUTSize/n (where n is the number of blocks and 4<=n) and BUTSize shall be 2 or more.
9	Configuration	Compliance with SW test strategy	Place resultTestRam1 and resultTestRam2 variables in different blocks of the RAM.

4.5 Directives for Software Configuration

Before compiling the code it is necessary to define the size of the RAM under test, the size of the memory block and the word length for the RAM test algorithm.
Table 4.4 shows how to define these information.

Table 4.4 Define directives

Directive	Description
MUTSizeX（X=1,2,3）	Size of the RAM under test. Set the size of RAM under test in 4-byte length units. Defined by the testRAM_config.h.
BUTSizeX（X=1,2,3）	RAM test block size. Value must be of the following type: MUTSize/4; MUTSize/5; MUTSize/6; MUTSize/7; MUTSize/8 ... ; MUTSize/MUTSize Set in 4-byte length units. Defined by the testRAM_config.h.
startAddressX（X=1,2,3）	Start address to the RAM area under test. Defined by the testRAM_config.h.

4.6 Software Package

Table 4.5 shows description of each design file.

Table 4.5 Design Files

#	File name	Description
1	testRAM.h	Declaration of RAM Test API (testRAM function to be called by the application SW)
2	testRAM.c	Definition of testRAM function.
3	testRAM.inc	Defines test execution pattern, start address used by .src file.
4	testRAM_config.h	Fixed address of test result variables and macro definition.
5	extendedMarchCminus.h	Declaration of Extended March C- algorithm function.
6	extendedMarchCminus.src	Definition of Extended March C- algorithm function.
7	extendedMarchCminus_forRambuffer.h	Declaration of Extended March C- algorithm function using RAM write buffer.
8	extendedMarchCminus_forRambuffer.src	Definition of Extended March C- algorithm function using RAM write buffer.
9	walpat.h	Declaration of WALPAT algorithm function.
10	walpat.src	Definition of WALPAT algorithm function.
11	walpat_forRambuffer.h	Declaration of WALPAT algorithm function using RAM write buffer.
12	walpat_forRambuffer.src	Definition of WALPAT algorithm function using RAM write buffer.

4.7 Resource Usage

Table 4.6 provides an overview of memory resources used by the code (little endian).
Maximum stack usage is 68 bytes.

Table 4.6 Memory Resources

Module	ROM (Byte)		RAM (Byte)
	Code	Data	
extendedMarchCminus.obj, extendedMarchCminus_forRambuffer.obj, testRAM.obj, walpat.obj, walpat_forRambuffer.obj	1,675	36	8

Table 4.7 shows execution time when RAM block size is 1 KByte.

Table 4.7 Execution Time

Algorithm	Non-destructive test Execution time [Clock cycles]	Non-destructive test Execution time @ 32 MHz clock [us]	Destructive test Execution time [Clock cycles]	Destructive test Execution time @ 32 MHz clock [us]
Extended March C-	83,465	2,608	81,314	2,541
WALPAT	6,600 ,290	206,259	6,598,157	206,192

4.8 Requirements for Safety Relevant Applications

Table 4.8 shows requirements of use in safety relevant applications.

Table 4.8 Requirements for Safety Relevant Applications

ID	Topic	Sub-topic	Description
RAM_SW_1	Test flow	Buffer	Perform destructive test on RAM block with buffer first. (Otherwise, a faulty buffer corrupts the data in the RAM block under test.)
RAM_SW_2	Configuration	Number of blocks	Number of memory block shall be as minimized as possible (4 blocks are preferred). (The larger the block size, more accurate the address fault detection)

5. Internal ROM Test

5.1 Test Objectives

The objective of ROM Test is to verify the internal ROM of the MCU.

Key features of the ROM Test are as follows,

- Whole memory test
- Block-wise test using multiple CRCs
- Supporting two CRC polynomials
- Incremental mode calculation: CRC signature is built up through multiple stages.

5.2 Test strategy

ROM Test searches faults from the internal ROM using CRC. Method of detecting error is as follows:

1. Specify the ROM address area: this step defines the block under test.
2. An expected checksum value is calculated using the linker and saved in the memory (See section 5.6).
3. Generate checksum of ROM area under test using CRC module during the ROM Test (See reference documents [7] - [22] for details of CRC module).
4. Compare the calculated checksum with the expected one. If values did not match, it is recognized as an error.
5. Repeat these steps until the whole ROM is covered.

5.2.1 Checksum Generation using the Tool Linker

Checksum generation by linker shall be enabled before compiling the ROM Test.

The items below also shall be confirmed.

1. Area addresses to store the checksum defined for each ROM area under test.
2. ROM address excluding checksum storage area
3. Checksum algorithm (Polynomial)

See section 5.6 for details.

5.2.2 CRC Module

The CRC module (refer to reference documents [7] - [22] for details) generates CRC codes for each data block. The following two polynomials are supported.

1. CRC-16 : $x^{16}+x^{15}+x^2+1$
2. CRC-CCITT : $x^{16}+x^{12}+x^5+1$

5.3 Software Structure

Two functions are available for checksum generation using CRC module.

- crcHwSetup: enables the CRC module and controls the CRC control register
- crcComputation: generates checksum for specified ROM block

5.3.1 ROM Test APIs

API functions are defined as follows:

void crcHwSetup(unsigned int crc)
uint16_t crcComputation(unsigned int checksumBegin, unsigned int checksumEnd, unsigned int crc, unsigned int incrMode)

Table 5.1 shows details of interface of the function.

Table 5.1 ROM Test APIs

#	Function	Type	C Type	Parameter	Description
1	crcHwSetup/ crcComputation	input	unsigned int	crc	CRC generating polynomial to use "1" : $x^{16}+x^{15}+x^2+1$ (CRC-16) "2" : $x^{16}+x^{12}+x^5+1$ (CRC-CCITT) Others : 16-bit CRC $x^{16}+x^{15}+x^2+1$
2	crcComputation	input	unsigned int	checksumBegin	ROM block start address
3	crcComputation	input	unsigned int	checksumEnd	ROM block end address
4	crcComputation	input	unsigned int	incrMode	CRC calculation mode "0" : Incremental mode not active Others : Incremental mode active
5	crcComputation	output	uint16_t	-	Generated checksum

Consider the information below for crcComputation function:

- CRC signature is initialized to the value below:
  - CRC-16 : 0x0000
  - CRC-CCITT : 0xffff
- the return value is
  - CRC-16 : calculated checksum
  - CRC-CCITT : 1's complement of the calculated checksum

Block size for CRC calculation can be figured out by the difference between the end and start address. Block size must be multiple of the CRC length.

5.3.2 Incremental Mode Calculation

The input parameter incrMode allows user to build up CRC signature value through multiple stages.

This method is summarized in Figure 5.1.

- Divide the target ROM block for CRC signature into multiple groups (3 in this case).
- Call the crcComputation function for each group.
- First group calls the crcComputation function with incremental mode disabled while the following groups call with incremental mode activated in order to "accumulate" previous results.
- After the final group, CRC of whole block is returned.

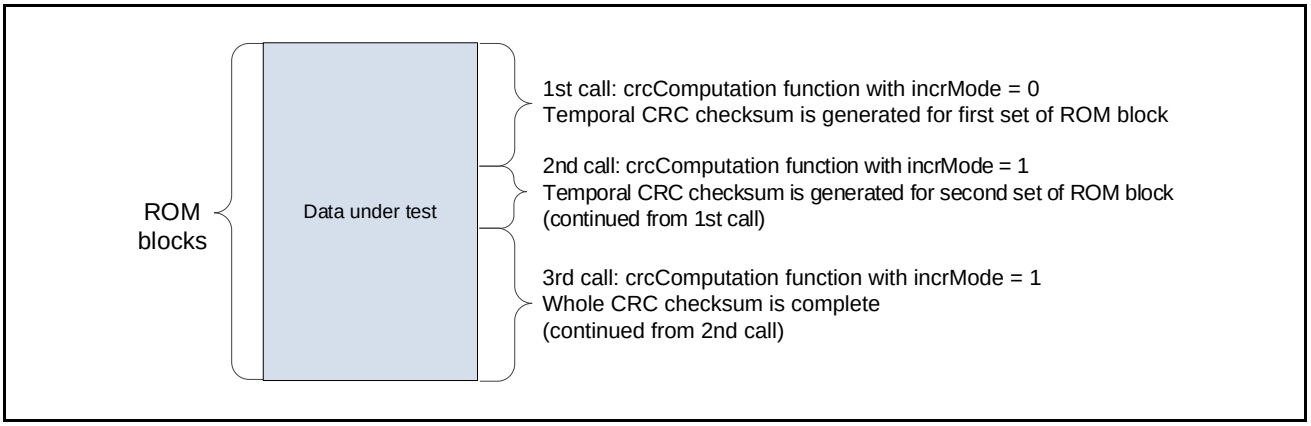

Figure 5.1 Incremental Mode Calculation

5.4 Software Integration Rules

5.4.1 Code Integration

Call the ROM Test as instructions below:

1. Include the testROM.h.
2. Define a pointer variable to store address of the CRC signature generated by linker.
3. Define variables for input parameter of crcHwSetup and crcComputation.
    a. *crcType*
    b. *checksumBegin*
    c. *checksumEnd*
    d. *incrMode*
4. Define output variable to store the result of the crcComputation.

See Section 5.4.2 for example of testing two ROM areas.

5.4.2 Test Flow and Test Results

A recommended flow of ROM Test is as below:

1. Initialize the CRC module using the crcHwSetup function.
2. Calculate the crcComputation function to calculate the checksum.

Compare the calculated checksum with the expected one.

Example:

```
#include "testROM.h"

uint16_t* _checksum1 = (uint16_t*)0xFFFFA000;

uint32_t type = 1;
crcHwSetup(type);

uint32_t checksumStart = 0xFFFFC000;
uint32_t checksumStop = 0xFFFFCFFF;
uint32_t crcIncr = 0;
uint16_t crcResult;

crcResult = crcComputation(checksumStart, checksumStop, type, crcIncr);
if(crcResult != *_checksum1){
   errorHandler();
}
```

In the example above, checks the result value to detect fault after the crcComputation function returns. (Compare the crcResult made during the ROM Test with the *_checksum1 made by linker)

Example of incremental mode activation

```
#include "testROM.h"

uint16_t* _checksum1 = (uint16_t*)0xFFFFA000;
uint32_t type;
uint32_t checksumStart;
uint32_t checksumStop;
uint16_t crcResult;
uint32_t crcIncr;

type = 1;
crcHwSetup(type);

crcIncr = 0;
checksumStart = 0xFFFFC000;
checksumStop = 0xFFFFC3FF;                    /* 1 KByte */
crcResult = crcComputation(checksumStart, checksumStop, type, crcIncr);

crcIncr = 1;
checksumStart = 0xFFFFC400;
checksumStop = 0xFFFFC7FF;                    /* 1 KByte    */
crcResult = crcComputation(checksumStart, checksumStop, type, crcIncr);

crcIncr = 1;
checksumStart = 0xFFFFC800;
checksumStop = 0xFFFFCBFF;                    /* 1 KByte */
crcResult = crcComputation(checksumStart, checksumStop, type,    crcIncr);

crcIncr = 1;
checksumStart = 0xFFFFCC00;
checksumStop = 0xFFFFCFFF;                    /* 1 KByte */

crcResult = crcComputation(checksumStart, checksumStop, type, crcIncr);
if(crcResult != *_checksum1){
   errorHandler();
}
```

NOTE: crcResult is compared with the expected value made by the linker after the crcComputation is called.

In the example above, the crcComputation function is called 4 times to make the CRC of 4-KByte ROM block.

5.4.3 Disabling ROM Cache Function

If you are using MCU with ROM cache, ROM cache function shall be disabled before the test.
See reference documents [7] - [22] for more information on the ROM cache.

5.4.4 Usage conditions

Table 5.2 summarizes usage conditions.

Table 5.2 Conditions of Use

ID	Topic	Condition	Description
1	Interrupt	Avoid corruption of function context	When interrupting the ROM Test, the context of all general purpose registers, system register including PSW, shall be saved and restored once returned from interrupt handling. See [5] [6] for details of the CPU register.
2	Interrupt	Avoid corruption of ROM cache status	For MCUs with ROM cache function, ROM cache shall be disabled before the test. When interrupting the ROM Test, do not change ROM cache settings. See reference documents [7] - [22] for details of the ROM cache function.
3	Incremental mode	Avoid corruption of the calculated CRC value.	In incremental mode, do not use the CRC module until the CRC calculation is complete.

5.5 Directives for Software Configuration

Table 5.3 shows directives to define settings around ROM cache function. This directive is defined in the testROM_config.inc. Customize the setting according to your environment.

Table 5.3 Directives for Software Configuration

Directive	Description
SYSTEM_PRCR	Address of Protest Register (PRCR)
SYSTEM_MSTPCRB	Address of Module Stop Control Register B (MSTPCRB)
CRC_CRCCR	Address of CRC Control Register (CRCCR)
CRC_CRCDIR	Address of CRC Data Input Register (CRCDIR)
CRC_CRCDOR	Address of CRC Data Output Register (CRCDOR)
CRC16_X16X15X2	Value to be set to CRCCR when 16-bit CRC ($X^{16} + X^{15} + X^2 + 1$) is used. Shall be set along with LSM bit = 0 and DORCLR bit = 1.
CRC16_X16X12X5	Value to be set to CRCCR when 16-bit CRC ($X^{16} + X^{15} + X^5 + 1$) is used. Shall be set along with LSM bit = 0 and DORCLR bit = 1.

5.6 Checksum Generation using the Tool

Expected checksum is required for the ROM Test, in order to compare with the checksum made by the CRC calculator.

This section shows how to generate the CRC using tools. The following information shall be provided to the linker.

1. Area address to store the calculated checksum
2. Start and end address of the ROM block under test
3. CRC type to be used
4. Endianness (big/little)

The above information can be provided with the following option:

-crc=<address where the result is output>=<start address>-<end address>/ <polynomial expression>: <endian>

- <address where the result is output>: Area address to store the calculated checksum
- <start address>-<end address>: Start and end address of the ROM under test (excluding checksum storage area)
- <polynomial expression>: CRC type (CCITT/16)
- <endian>: Endianness (big/little)

NOTE: Address to store the calculated CRC is not subject to the ROM Test.

Add the CRC option by selecting CRC operation option from the following property.

Project > C/C++ Project Settings > Tool Settings > Converter > CRC Operation

Checksum Generation Example

The following address is tested.

- 0xFFFFC000- 0xFFFFCFFF

When the checksum is produced by the polynomial $x^{16}+x^{15}+ x^2+1$ (CRC-16) and the expected value is stored to the address 0xFFFFA000, the definition is as shown below:

- Outputs the calculation result of CRC (-crc): Ticked
- Output address (-crc(Output address)): FFFFA000
- Type of CRC (-crc (Type)): 16
- Endian and Output size (-crc (Endian and Output size)): LITTLE
- Target range (-crc (Target range)): FFFFC000-FFFFCFFF

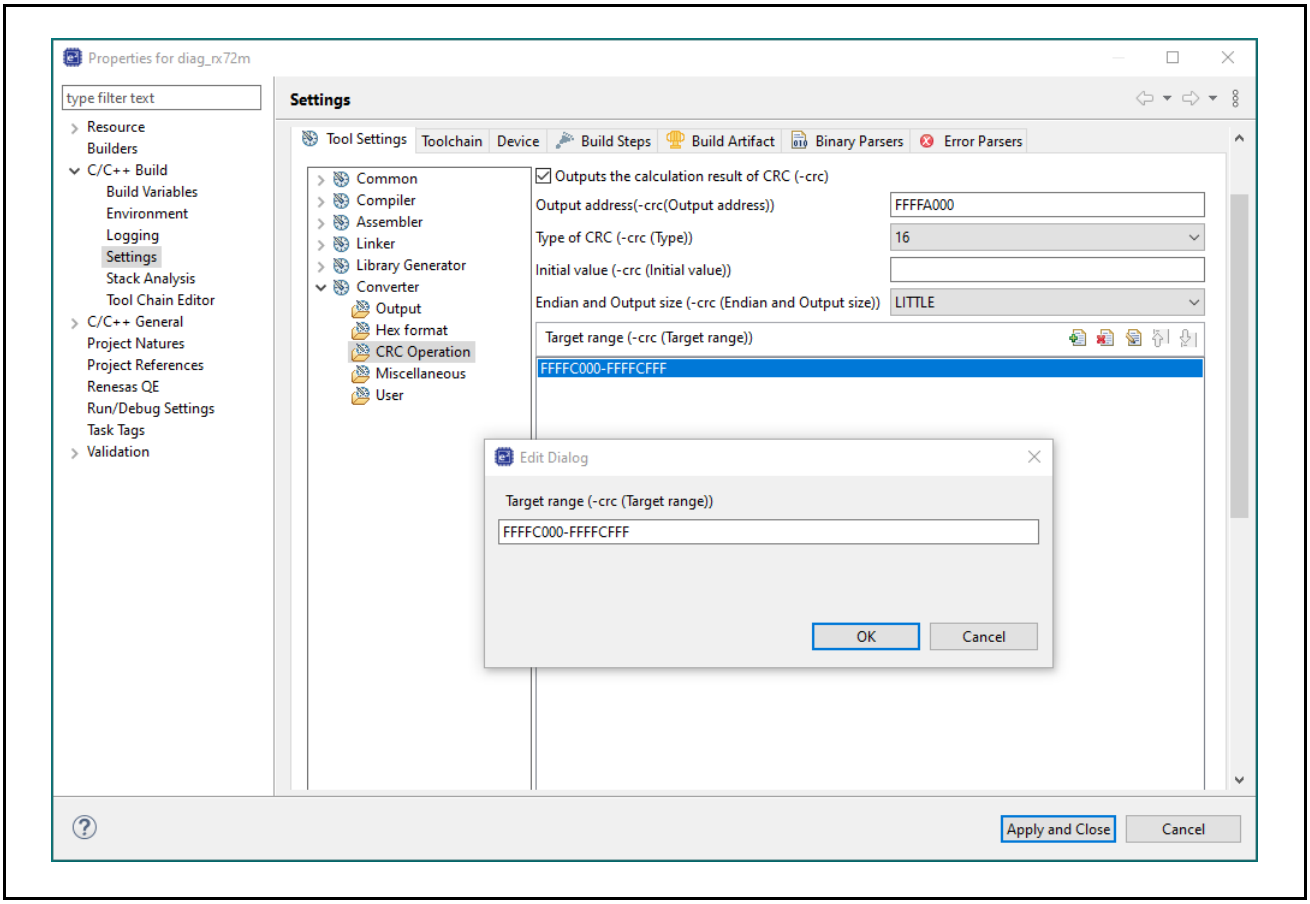

Figure 5.2 Environment Options

See reference document [4] for details about the CRC option.

5.7 Software Package

Table 5.4 shows description of each design file.

Table 5.4 Design Files

#	File name	Description
1	crc.src	Definition of the functions declared in the testROM.h
2	testROM.h	Declaration of the crcHwSetup and crcComputation functions crcHwSetup : initialization of CRC module crcComputation : Runs CRC on the specified ROM block
3	testROM_config.inc	Setting related to MCU ROM cache for assembler

5.8 Resource Usage

Table 5.5 provides an overview of the memory resource used by the code (little-endian)
Maximum stack usage is 0 bytes.

Table 5.5 Memory Resources

Module	ROM (Byte)		RAM (Byte)
	Code	Data	
crc.obj	107	0	0

Table 5.6 illustrates the execution time of CRC calculator for 4-KByte ROM block when the polynomial $x^{16}+x^{15}+x^2+1$ is used. Execution time is virtually the same when the other polynomial is used.

Table 5.6 Execution time

Function	Execution time when ROM block size = 4 KBytes [Clock cycles]	Execution time when ROM block size = 4 KBytes @32MHz clock [us]
crcComputation	34,234	1,070

5.9 Requirements for Safety Relevant Applications

Table 5.7 shows requirements of use in safety relevant applications.

Table 5.7 Requirements for Safety Relevant Applications

ID	Topic	Sub-topic	Description
ROM_SW_1	CRC type	-	CRC polynomial $x^{16}+x^{15}+x^2+1$ is used
ROM_SW_2	Block length	-	Block size = 4 KBytes

By following the above requirements, all single-and-double bit corruptions can be detected. It also enables you to detect odd number of single bit error, regardless of the block size. burst of length k corresponds to the presence of k consecutive corrupted bits:

- all bursts with length equal and less than 16 bits
- 99.997 percent of bursts of 17 bits
- 99.998 percent of bursts with length greater than 18 bits.

6. Appendix

6.1 Appendix A - CPU Test Signature Values

Table 6.1 - Table 6.9 shows the expected signature values of each R_CPU_Diag functions used in CPU Test.
Table 6.1 and Table 6.2 list expected signature values of R_CPU_Diag functions of type Fixed. Expected values for little and big endian are stated when the dependency of endianness is relevant. Blank columns in the table indicate that the expected values are the same as Little Endian.
Table 6.3 - Table 6.8 list expected signature values of R_CPU_Diag functions of type LUT. Expected values have no dependency on endianness here.
Table 6.9 lists expected signature values of R_CPU_Diag functions of type Specific. Expected values have no dependency on endianness here too.

Table 6.1 Expected Signature Values of "Fixed" R_CPU_Diag Functions (1 of 2)

Test #	Signature Little endian	Neg-Signature Little endian	Signature Big endian	Neg-Signature Big endian
0	0xB3B83757	0x4C47C8A9		
1	0xCCCCCCCB	0x33333335		
2	0x0EAEED19	0xF15112E7	0x49AEED5E	0xB65112A2
3	0x0A6C01E7	0xF593FE19		
4	0x4B7E308E	0xB481CF72		
5	0xACAC0563	0x5353FA9D		
6	0x03550DA5	0xFCAAF25B		
7	0x62711A01	0x9D8EE5FF	0x42EBB8AB	0xBD144755
8	0xC5BBA3DF	0x3A445C21		
9	0x5579550D	0xAA86AAF3		
10	0xD555555A	0x2AAAAAA6		
11	0x42660755	0xBD99F8AB		
12	0x25992687	0xDA66D979		
13	0xB649C8F6	0x49B6370A		
14	0x97AE0293	0x6851FD6D		
15	0x059C4813	0xFA63B7ED		
16	0x3B9251AD	0xC46DAE53		
17	0x565D6936	0xA9A296CA		
18	0xD1AAD75E	0x2E5528A2		
19	0x41C70017	0xBE38FFE9		
20	0xAD5D81A5	0x52A27E5B	0xA5E45CA4	0x5A1BA35C
21	0x55AA5556	0xAA55AAAA		
22	0x05F59BB9	0xFA0A6447	0x04128432	0xFBED7BCE
23	0x54444644	0xABBBB9BC		

Table 6.2 Expected Signature Values of "Fixed" R_CPU_Diag Functions (2 of 2)

Test #	Signature Little endian	Neg-Signature Little endian	Signature Big endian	Neg-Signature Big endian
24 (RX72M,RX72N, RX72T, RX66T, RX66N, RX660, RX671)	0xB6669F67	0x49996099	0xFDCCB5CA	0x02334A36
24 (RX71M, RX64M, RX65N, RX651, RX200 series)	0x14442BD1	0xEBBBD42F	0x5FEE017C	0xA011FE84
25	0x6AAB6AD5	0x9554952B		
26	0x7FFF8003	0x80007FFD		
27	0xEF88888E	0x10777772		
28	0x57739835	0xA88C67CB		
29	0x1C40C15A	0xE3BF3EA6		
30	0x65C3DD1F	0x9A3C22E1		
31	0x4AA576AA	0xB55A8956		
32	0xB4AA30AC	0x4B55CF54		
33	0xAAAAAAEA	0x55555516		
34	0xAAAAA96A	0x55555696		
35	0xAAAAAA1A	0x555555E6		
36	0xAAAAAAAA	0x55555556		
37	0xAAAAAAAA	0x55555556		
38	0xAAAAAAAA	0x55555556		
39	0xAAAAAAAA	0x55555556		
40	0xAAAAAAAA	0x55555556		
41	0xAAAAAAAA	0x55555556		

Table 6.3 Expected Signature Values of "LUT" R_CPU_Diag Functions (1 of 6)

Test #	Data Set #	Signature (All endianness)	Neg-Signature (All endianness)
50	0	0x1B01CCCA	0xE4FE3336
50	1	0x80D92A1E	0x7F26D5E2
50	2	0xB4E7293D	0x4B18D6C3
50	3	0xFBFDE16F	0x04021E91
50	4	0xB4F4926D	0x4B0B6D93
50	5	0x7E87E9A8	0x81781658
51	0	0x8BEC8395	0x74137C6B
51	1	0x42EB9D81	0xBD14627F
51	2	0xFD90032A	0x026FFCD6
51	3	0xC0962E04	0x3F69D1FC
51	4	0x0633A473	0xF9CC5B8D
51	5	0x48F8DEB5	0xB707214B
52	0	0x2FC552DB	0xD03AAD25
52	1	0xD9C19E2F	0x263E61D1
52	2	0xE3739715	0x1C8C68EB
52	3	0x3BF1E6F3	0xC40E190D
52	4	0xBF2F6E03	0x40D091FD
52	5	0x0325E92B	0xFCDA16D5
53	0	0x9D94C9A2	0x626B365E
53	1	0x89D40D5F	0x762BF2A1
53	2	0xF7F9682B	0x080697D5
53	3	0x21926923	0xDE6D96DD
53	4	0x2E37A794	0xD1C8586C
53	5	0x557C9893	0xAA83676D
54	0	0xEBB272FA	0x144D8D06
54	1	0x5EEF66C0	0xA1109940
54	2	0xCE5E3F23	0x31A1C0DD
54	3	0xC1165A12	0x3EE9A5EE
54	4	0x8EF451FF	0x710BAE01
54	5	0x4E97C468	0xB1683B98
55	0	0xE3D9A92D	0x1C2656D3
55	1	0xA8BDF2F4	0x57420D0C
55	2	0x0568F838	0xFA9707C8
55	3	0xD6481821	0x29B7E7DF
55	4	0xCFF20DC1	0x300DF23F

Table 6.4 Expected Signature Values of "LUT" R_CPU_Diag Functions (2 of 6)

Test #	Data Set #	Signature (All endianness)	Neg-Signature (All endianness)
55	5	0x0790CFEA	0xF86F3016
56	0	0xFC7F3545	0x0380CABB
56	1	0x9263C466	0x6D9C3B9A
56	2	0x1DD61F66	0xE229E09A
56	3	0xCA31973D	0x35CE68C3
56	4	0x646AAF50	0x9B9550B0
56	5	0x89391DE2	0x76C6E21E
57	0	0x98519BF7	0x67AE6409
57	1	0x6317C899	0x9CE83767
57	2	0xC89FCE9B	0x37603165
57	3	0x41058965	0xBEFA769B
57	4	0x528DB325	0xAD724CDB
58	0	0x2E0AF46A	0xD1F50B96
58	1	0xD9EB78D6	0x2614872A
58	2	0xE44FE363	0x1BB01C9D
58	3	0x5F7D3B4D	0xA082C4B3
58	4	0xA9474EB4	0x56B8B14C
59	0	0xA6A736D8	0x5958C928
59	1	0x378816CB	0xC877E935
59	2	0xB680B65C	0x497F49A4
59	3	0x2A9791D9	0xD5686E27
59	4	0x1CB285B0	0xE34D7A50
59	5	0xB4199E50	0x4BE661B0
59	6	0x05A955D4	0xFA56AA2C
59	7	0xD666710E	0x29998EF2
59	8	0x14461B2C	0xEBB9E4D4
59	9	0xD1442B80	0x2EBBD480
59	10	0x19F4D2E7	0xE60B2D19
59	11	0x93172363	0x6CE8DC9D
59	12	0xD52AAAA8	0x2AD55558
59	13	0x542AAAAF	0xABD55551
59	14	0xAAAAAAAB	0x55555555
59	15	0x5B8430E1	0xA47BCF1F
60	0	0x66B72E16	0x9948D1EA
60	1	0xB78B61CF	0x48749E31

Table 6.5 Expected Signature Values of "LUT" R_CPU_Diag Functions (3 of 6)

Test #	Data Set #	Signature (All endianness)	Neg-Signature (All endianness)
60	2	0x7788CFB5	0x8877304B
60	3	0x944D7812	0x6BB287EE
60	4	0x1CDD5C2B	0xE322A3D5
60	5	0x3416C0D1	0xCBE93F2F
60	6	0x04635463	0xFB9CAB9D
60	7	0xD659C01C	0x29A63FE4
60	8	0x1444EFC1	0xEBBB103F
60	9	0xD0941FF4	0x2F6BE00C
60	10	0x198F5D65	0xE670A29B
60	11	0x93025853	0x6CFDA7AD
60	12	0x552AAAAC	0xAAD55554
60	13	0xAAAAAAAB	0x55555555
60	14	0x2AAAAAAF	0xD5555551
60	15	0xA57BCF72	0x5A84308E
61	0	0x6221DC50	0x9DDE23B0
61	1	0x58CFCC08	0xA73033F8
61	2	0xD99CE9D6	0x2663162A
61	3	0x038F4CED	0xFC70B313
61	4	0x1EA188B2	0xE15E774E
61	5	0xA3DEEB3D	0x5C2114C3
61	6	0xFA7C03E0	0x0583FC20
61	7	0x9722D9BF	0x68DD2641
61	8	0x61F75EA1	0x9E08A15F
61	9	0x7421E612	0x8BDE19EE
61	10	0x6C7B48C2	0x9384B73E
61	11	0xEE1C5059	0x11E3AFA7
61	12	0x2AAAAAAC	0xD5555554
61	13	0xEAAAAAAB	0x15555555
61	14	0x2AAAAAAF	0xD5555551
61	15	0xF51B3A8D	0x0AE4C573
62	0	0xC857340A	0x37A8CBF6
62	1	0x2715AFCD	0xD8EA5033
62	2	0x18F8D089	0xE7072F77
62	3	0x5B6FE7CF	0xA4901831
62	4	0x2A7F6BBB	0xD5809445

Table 6.6 Expected Signature Values of "LUT" R_CPU_Diag Functions (4 of 6)

Test #	Data Set #	Signature (All endianness)	Neg-Signature (All endianness)
62	5	0xBA4996AF	0x45B66951
62	6	0xD767A935	0x289856CB
62	7	0xD3D66B85	0x2C29947B
62	8	0xD6A4AEE8	0x295B5118
62	9	0x51FD65AA	0xAE029A56
62	10	0x33D83F6B	0xCC27C095
62	11	0x0D984B26	0xF267B4DA
62	12	0x51555551	0xAEAAAAAF
62	13	0xAAAAAAAB	0x55555555
62	14	0xAAAAAAAB	0x55555555
62	15	0x4C89055B	0xB376FAA5
63	0	0xBA2BAA2D	0x45D455D3
64	0	0x85D455DB	0x7A2BAA25
65	0	0xB44EBCB5	0x4BB1434B
65	1	0xB99B960F	0x466469F1
65	2	0xB8EEFE97	0x47110169
65	3	0xAC3EA7A9	0x53C15857
65	4	0xD302E9C6	0x2CFD163A
65	5	0xACA1C7ED	0x535E3813
65	6	0xD3D786F2	0x2C28790E
65	7	0xC71115D3	0x38EEEA2D
65	8	0xC6064BFE	0x39F9B402
65	9	0xCBE8BCD8	0x34174328
66	0	0x0BE21D7B	0xF41DE285
66	1	0xB7CE91A4	0x48316E5C
67	0	0xAAAAAAAA	0x55555556
67	1	0xAAAAAAAA	0x55555556
67	2	0xAAAAAAAA	0x55555556
67	3	0xAAAAAAAA	0x55555556
67	4	0xAAAAAAAA	0x55555556
67	5	0xAAAAAAAA	0x55555556
67	6	0xAAAA9AAA	0x55556556
67	7	0xB52AB579	0x4AD54A87
68	0	0xAAA837AA	0x5557C856
68	1	0xAAA711AA	0x5558EE56

Table 6.7 Expected Signature Values of "LUT" R_CPU_Diag Functions (5 of 6)

Test #	Data Set #	Signature (All endianness)	Neg-Signature (All endianness)
68	2	0xAAACB7AA	0x55534856
68	3	0xAAAB15AA	0x5554EA56
68	4	0xAAA80FAA	0x5557F056
68	5	0x643F05FF	0x9BC0FA01
68	6	0xE4B55D0A	0x1B4AA2F6
68	7	0x170BAB41	0xE8F454BF
68	8	0x9780B4EA	0x687F4B16
68	9	0x06074A3F	0xF9F8B5C1
68	10	0x8685B24A	0x797A4DB6
68	11	0x311A8B81	0xCEE5747F
68	12	0xB191F72A	0x4E6E08D6
68	13	0x2017897F	0xDFE87681
69	0	0x35FB2CF4	0xCA04D30C
69	1	0xB87B0C73	0x4784F38D
69	2	0x38FE79A2	0xC701865E
69	3	0xAF7E5924	0x5081A6DC
69	4	0x2FF28A6E	0xD00D7592
69	5	0xD2736DE8	0x2D8C9218
69	6	0x52F34D5A	0xAD0CB2A6
69	7	0xAAAAAAD3	0x5555552D
69	8	0xAAAAAA9C	0x55555564
69	9	0x4AF55504	0xB50AAAFC
70	0	0xAAAAA8B9	0x55555747
70	1	0xAAAAABE0	0x55555420
70	2	0xAAAAABE3	0x5555541D
70	3	0xAAAAA8B8	0x55555748
71	0	0x3D432836	0xC2BCD7CA
71	1	0x7FDFB63A	0x802049C6
71	2	0x5372CD99	0xAC8D3267
71	3	0x5E30746A	0xA1CF8B96
71	4	0xB42488FA	0x4BDB7706
71	5	0xA2CF6BBA	0x5D309446
72	0	0x5A5A5A5A	0xA5A5A5A5
72	1	0xFFFFFFFF	0x00000000
72	2	0x00000001	0xFFFFFFFE

Table 6.8 Expected Signature Values of "LUT" R_CPU_Diag Functions (6 of 6)

Test #	Data Set #	Signature (All endianness)	Neg-Signature (All endianness)
72	3	0xA5A5A5A5	0x5A5A5A5A
72	4	0x5A5A5A5A	0xA5A5A5A5
72	5	0xFFFFFFFF	0x00000000
72	6	0x00000002	0xFFFFFFFD
72	7	0xA5A5A5A5	0x5A5A5A5A
72	8	0x5A5A5A5A	0xA5A5A5A5
72	9	0xFFFFFFFF	0x00000000
72	10	0x80000001	0x7FFFFFFE
72	11	0xA5A5A5A5	0x5A5A5A5A
72	12	0x5A5A5A5A	0xA5A5A5A5
72	13	0xFFFFFFFF	0x00000000
72	14	0x40000000	0xBFFFFFFF
73	0	0x5A5A5A5A	0xA5A5A5A5
73	1	0xFFFFFFFF	0x00000000
73	2	0x00000001	0xFFFFFFFE
73	3	0xA5A5A5A5	0x5A5A5A5A
73	4	0x5A5A5A5A	0xA5A5A5A5
73	5	0xFFFFFFFF	0x00000000
73	6	0x00000002	0xFFFFFFFD
73	7	0xA5A5A5A5	0x5A5A5A5A
73	8	0x5A5A5A5A	0xA5A5A5A5
73	9	0xFFFFFFFF	0x00000000
73	10	0x80000001	0x7FFFFFFE
73	11	0xA5A5A5A5	0x5A5A5A5A
73	12	0x5A5A5A5A	0xA5A5A5A5
73	13	0xFFFFFFFF	0x00000000
73	14	0x40000000	0xBFFFFFFF

Table 6.9 Expected Signature Values of "Specific" R_CPU_Diag Functions

Test #	Signature (All endianness)	Neg-Signature (All endianness)
100	0x23DDDDC2	0xDC22223E
101	0x555D750E	0xAAA28AF2
102	0xAAAAAAC8	0x55555538
103 (RX72M, RX72N, RX66N, RX671)	0xAAAA94CD	0x55556B33
103 (RX71M, RX72T, RX64M, RX65N, RX651, RX66T, RX660, RX200 series)	0xAAAAD6CD	0x55552933
104	0xE5D55511	0x1A2AAAEF

6.2 Annex B - RAM Test Algorithms

Table 6.10 shows characteristics of two test algorithms available for the RAM Test.

Table 6.10 RAM Test Algorithm Characteristics

Fault models and complexity	Extended March C-	WALPAT
Address Faults (AF)	✓	✓
Stuck At faults (SAF)	✓	✓
Transactional Faults (TF)	✓	✓
Coupling Faults (CF)	✓	✓
Stuck-Open Faults (SOF)	✓	N/A
Data Retention Faults (DRF)	✓	N/A
Sense Amplifier Recovery Faults (SARF)	N/A	✓
Complexity *1	11n	$2n^2$

*1: n = Number of cells in the memory

The following algorithm descriptions are related to 1-bit memory, but they can be applied to m-bit memories. m-bit memories can be dealt with by repeating each algorithm for several times determined by:

$$\lceil \log_2 m \rceil + 1$$

Since m=32bit for this software, the algorithm will be repeated 6 times and the following 6 different patterns have to be applied.

#1: 00000000000000000000000000000000
#2: 00000000000000001111111111111111
#3: 00000000111111110000000011111111
#4: 00001111000011110000111100001111
#5: 00110011001100110011001100110011
#6: 01010101010101010101010101010101

6.2.1 Extended March C-

Extended March C- is an March test algorithm used for RAM testing.

Extended March C- algorithm is represented in Figure 6.1.

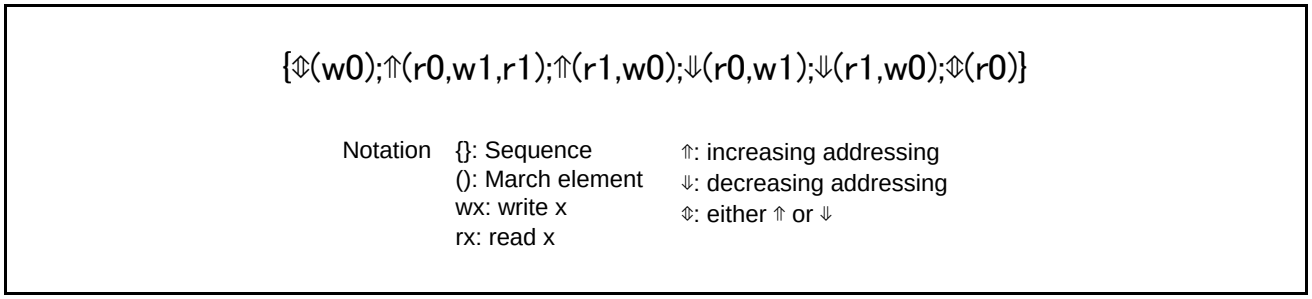

Figure 6.1 Extended March C- Algorithm

6.2.2 WALPAT

WALPAT (stands for Walking Pattern) is another algorithm used for RAM testing.

WALPAT algorithm is represented in Figure 6.2.

```
Write 0 in all cells;
For i=0 to n-1
{
    complement cell[i];
    For j=0 to n-1, j != i
    {
        read cell[j];
    }
    read cell[i];
    complement cell[i];
}
Write 1 in all cells;
For i=0 to n-1
{
    complement cell[i];
    For j=0 to n-1, j != i
    {
        read cell[j];
    }
    read cell[i];
    complement cell[i];
}
```

Figure 6.2 WALPAT Algorithm

6.3 Appendix C - CPU Test Example

```
#include <machine.h>
#include "testCPU.h"

int32_t signature[2];

const int32_t offsetSig[24] =
{
    0, (OffsetDS51 - 1), (OffsetDS52 - 2), (OffsetDS53 - 3), (OffsetDS54 - 4), (OffsetDS55 - 5),
    (OffsetDS56 - 6), (OffsetDS57 - 7), (OffsetDS58 - 8), (OffsetDS59 - 9), (OffsetDS60 - 10),
    (OffsetDS61 - 11), (OffsetDS62 - 12), (OffsetDS63 - 13), (OffsetDS64 - 14), (OffsetDS65 - 15),
    (OffsetDS66 - 16), (OffsetDS67 - 17), (OffsetDS68 - 18), (OffsetDS69 - 19), (OffsetDS70 - 20),
    (OffsetDS71 - 21), (OffsetDS72 - 22), (OffsetDS73 - 23),
};

const int32_t numDS[24] =
{
    numDataSet50, numDataSet51, numDataSet52, numDataSet53, numDataSet54, numDataSet55,
    numDataSet56, numDataSet57, numDataSet58, numDataSet59, numDataSet60, numDataSet61,
    numDataSet62, numDataSet63, numDataSet64, numDataSet65, numDataSet66, numDataSet67,
    numDataSet68, numDataSet69, numDataSet70, numDataSet71, numDataSet72, numDataSet73,
};

const int32_t signatureRefNorm[42][2] =
{
    /* Expected Signature for Fixed type function */
};

const int32_t signatureRefTbl[202][2] =
{
    /* Expected Signature for LUT type function with all data set */
};

const int32_t signatureRefSpec[5][2] =
{
    /* Expected Signature for Specific type function */
};

int32_t result = 0;
const uint32_t faultInj = 11;

void Test_Passed(void)
{
    /* Test passed */
}

void errorHandler(void)
{
    /* Error Handler */
}
```

```
void main(void)
{
    uint32_t reg;
    uint32_t i; /* index */
    uint32_t dataSet;

    /* disable interrupt */
    reg = get_psw();
    if (0x00020000 == (reg & 0x00020000))
    {
        errorHandler();
    }
    reg &= 0xFFFEFFFF;    /* Clear I bit, disable interrupt */
    set_psw(reg);

    /* Table type function */
    for (i = 0; i < 24; i++)
    {
        for (dataSet = 0;dataSet < numDS[i]; dataSet++)
        {
            result = 0;
            R_CPU_DiagTbl(i, dataSet, faultInj, signature, &result);

            if ((result != 1) || (signature[0] != signatureRefTbl[i + dataSet + offsetSig[i]][0])
                || (signature[1] != signatureRefTbl[i + dataSet + offsetSig[i]][1]))
            {
                errorHandler();
            }
        }
    }

    /* Normal type function */
    for (i = 0; i < 42; i++)
    {
        result = 0;
        R_CPU_DiagNorm(i, faultInj, signature, &result);

        if ((result != 1) || (signature[0] != signatureRefNorm[i][0])
            || (signature[1] != signatureRefNorm[i][1]))
        {
            errorHandler();
        }
    }
```

```
    /* Initalize for R_CPU_Diag100 */
    R_CPU_Diag_IntControl(&result, 1);
    if (result != 1)
    {
        errorHandler();
    }

    /* Specific type function (interrupt, Exception) */
    for (i = 0; i < 5; i++)
    {
        result = 0;
        R_CPU_DiagSpec(i, faultInj, signature, &result);

        if ((result != 1) || (signature[0] != signatureRefSpec[i][0]) || (signature[1] != signatureRefSpec[i][1]))
        {
            errorHandler();
        }
    }

    Test_Passed();
}
```

6.4 Appendix D - Notes on Internal ROM Test

6.4.1 Option Function Select Register 1 (OFS1) Setting

When development tool is started up to initiate FINE communication, the emulator forcibly sets bits 25 and 24 in the OFS1 register to 10b. Then ROM Test is fails in the ROM area including OFS1 register. See [7] - [22] for the address of the OFS1 register.

6.4.2 Placement of Reference Checksum

When Reference checksum is stored in ROM area, please consider the following:

- Do not store reference checksum in area that contains the exception vector table
  See [7] - [22] for details about the exception vector table.

6.4.3 Recommendation on Placement of Reference Checksum

To place reference checksum, there are following recommendations:

a) Set start address or end address of each ROM block calculating checksum.

b) Place each reference checksum into one ROM area.

When internal ROM Test is used according to recommendations for usage in safety relevant applications, please select b).

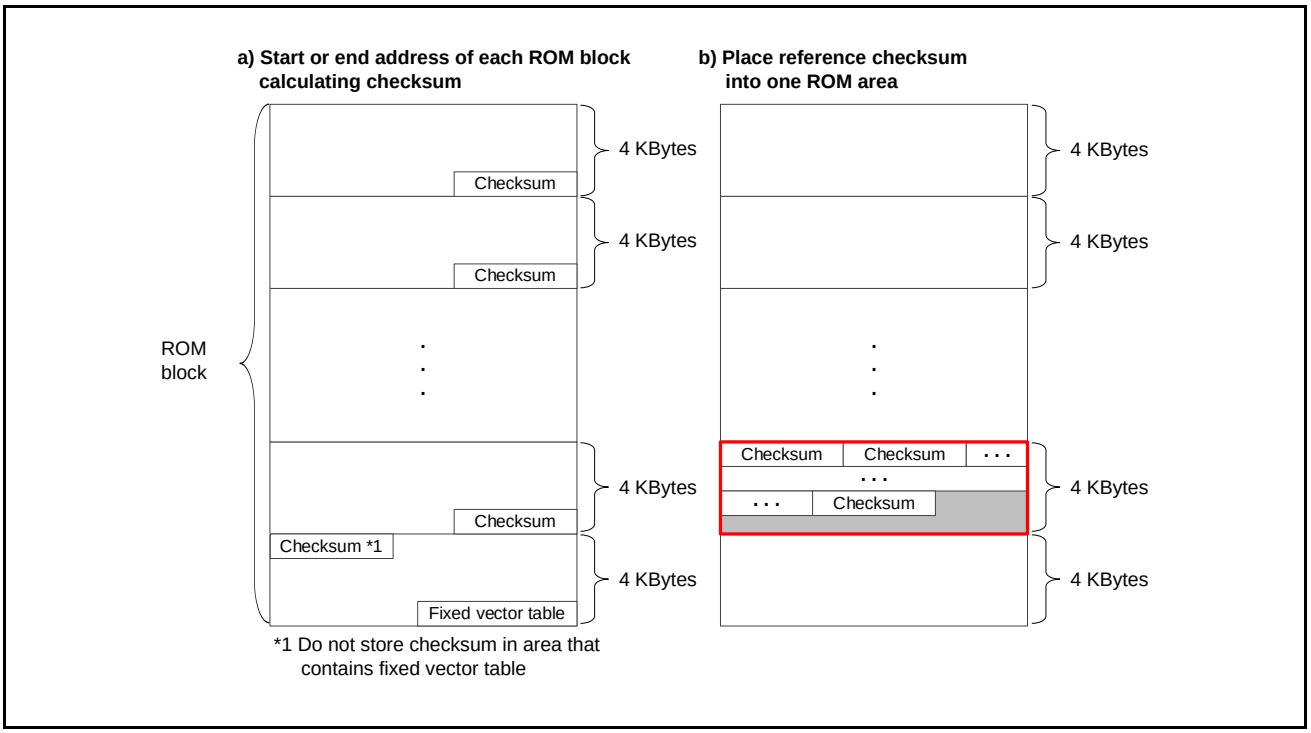

Figure 6.3 Placement of Reference Checksum

6.5 Appendix E - MD5 Checksum

6.5.1 Identification and Contents of Package of CPU Test

CPU core test package version is identified as follows:

- Version 3.00

Table 6.11 CPU core test package and MD5 signatures (1 of 2)

File name	MD5 checksum	File name	MD5 checksum
testCPU.h	de8efcabc92c550dcaf3e57707cd9c03	r_cpu_diag_11_rx.src	891ad8d5f586895c8c442665d0a8005d
r_cpu_diag_rx.c	173843b7ae961342a2a49bddb1334ed6	r_cpu_diag_12_rx.h	38870d5c18b336e2b29c5fffa1716f0f
testCPU_config.h	e14412a0fa1187080628ed3205b875f0	r_cpu_diag_12_rx.src	e6a0c1549c4ed9e4f96f6e57045c8643
testCPU_config.inc	1a47b8a1ba241d6ee5841e7b62a0221e	r_cpu_diag_13_rx.h	f04f396d402db7875855744a0fcb6a04
r_cpu_diag_rx_private.h	039717adb691da2aed791749df6072de	r_cpu_diag_13_rx.src	060ed796d5bf835c460a73ba3b150b3f
globVarAsm.inc	5c6f82fcce5871724629bcd434a8adf7	r_cpu_diag_14_rx.h	ca9539b68de93a1328915f4b8108b873
intCtrl.h	dc9304be3de582af4eb2d426a7c1d374	r_cpu_diag_14_rx.src	08aafc200b1ba503ca393ddfb09fce22
intCtrl.src	662a211d6ea7349af13dee8490e456e8	r_cpu_diag_15_rx.h	c19615e6170a41faaa723e05e1fdaab0
romCacheCtrl.h	8206a4b40773054861bf4a6aac46bf7c	r_cpu_diag_15_rx.src	ed5987d02d7bce4a8ce3f0ab69f8300a
romCacheCtrl.src	8ff8ee4c8c20811d7b04b33795d1ccd4	r_cpu_diag_16_rx.h	a3ec4ef1347a0a6ab893c87863088ba0
_signatureCrt.src	d8cb774bd5d0002d51330e1432df67fc	r_cpu_diag_16_rx.src	4980689115ddc7130a30cffae6a660c7
r_cpu_diag_0_rx.h	28aafaea3e5d0f8eebd1675fc891afaf	r_cpu_diag_17_rx.h	05b5b910a33efcca09fe525b5b88225e
r_cpu_diag_0_rx.src	1b7c00b88e0dc6e3815940aab2a47a45	r_cpu_diag_17_rx.src	923a93ad4239fe815d72d22316ae1069
r_cpu_diag_1_rx.h	26b13a6c67d936b9e3eb3e899b452b11	r_cpu_diag_18_rx.h	33968f6bedd5da362eb752e57f260c0c
r_cpu_diag_1_rx.src	7b8d0c5e2f33298e2b8c43c84ad62a24	r_cpu_diag_18_rx.src	a09698859cdb34b1dcd597ca0198a20e
r_cpu_diag_2_rx.h	4e601925881142095931 77e545193576	r_cpu_diag_19_rx.h	eb73a3dd96e049b1fc2ed55f22ac9be2
r_cpu_diag_2_rx.src	185dd2b99bb09a88f78362ced8e6b942	r_cpu_diag_19_rx.src	47fea9cd4f685e9677e59382dd2c65db
r_cpu_diag_3_rx.h	e4cc7811c81ace202b1c55a65fcb4b1e	r_cpu_diag_20_rx.h	ee2c46e3470d2a4935953d1367ce7fef
r_cpu_diag_3_rx.src	2337994bfca46f7aa1800d511f1a1e70	r_cpu_diag_20_rx.src	dbee64f021c5c769fd74c98474b138f5
r_cpu_diag_4_rx.h	dc6300d1e4119a981cfd7123d795adc0	r_cpu_diag_21_rx.h	9d5504e151bda45fe5297c1cf8595a59
r_cpu_diag_4_rx.src	d7e18b74d85432531826d9a1db8d18af	r_cpu_diag_21_rx.src	81345dadcff6b63048b201a156b72b55
r_cpu_diag_5_rx.h	3234f0306eeb571a287b3410fc78bf50	r_cpu_diag_22_rx.h	13006f0a3786016269766a268625a6fd
r_cpu_diag_5_rx.src	efe4679108e6f6181b918162be19b3ec	r_cpu_diag_22_rx.src	11f3ed989da1dc2101a792ae0b034e33
r_cpu_diag_6_rx.h	ff1ed9416a505165300fc8fc499b2aa4	r_cpu_diag_23_rx.h	12ef7257a607834076dd500cff9262b4
r_cpu_diag_6_rx.src	337245b1e2a45994550113cb05bfc578	r_cpu_diag_23_rx.src	3e8b38e43f18d72c1e680772fc918f67
r_cpu_diag_7_rx.h	a2072bb1136741a06908f41f85b7af4b	r_cpu_diag_24_rx.h	e051822fadcda41fa8c053bfed6aad91
r_cpu_diag_7_rx.src	8744cc5960c3b523a4b73071efeffa63	r_cpu_diag_24_rx.src	de8a2d9c6f1c96b4fb246c0c30e26107
r_cpu_diag_8_rx.h	21804bae25398d36ca69159e5bf609ef	r_cpu_diag_25_rx.h	f2b047260e1dbca089ea2c12aac23733
r_cpu_diag_8_rx.src	77e9f66cb195aea660ad668b93800e91	r_cpu_diag_25_rx.src	d0ad6994e1fc84bd3376393ac1bcd552
r_cpu_diag_9_rx.h	25bf6d68e2964557ab3e5c41e706233f	r_cpu_diag_26_rx.h	53fb5cdf6bd3eef257c6fa32d63d12a0
r_cpu_diag_9_rx.src	39dcebe9e1ed1a2bd048d93341d3d6a4	r_cpu_diag_26_rx.src	501c538bffe1ec6336309724a46e2d4f
r_cpu_diag_10_rx.h	83a89ec39cf260b7018ba2ef61b03245	r_cpu_diag_27_rx.h	3055780e5a638b30fe2305c609ffcf14
r_cpu_diag_10_rx.src	ba662a7d3cdbbbf76b9b2ddfaf8df367	r_cpu_diag_27_rx.src	f4ca1099fe26c92d867ec8e33b4a8495
r_cpu_diag_11_rx.h	6dc2a38758643a0a9e61566a14869bf9	r_cpu_diag_28_rx.h	e8c59927706cf5d55c2cb8de7ff8ab84

Table 6.12 CPU core test package and MD5 signatures (2 of 2)

File name	MD5 checksum	File name	MD5 checksum
r_cpu_diag_28_rx.src	6e1413d87ed632e45057d710d2615401	r_cpu_diag_58_rx.h	cbe80b6258f812b1c14bba5ee2323d21
r_cpu_diag_29_rx.h	788bf17e6bf705f4025e019f15207b5f	r_cpu_diag_58_rx.src	03787d0bc1ab96d4f1222ac153b30bc7
r_cpu_diag_29_rx.src	9083f77dc64497916aa088ab9c7eee65	r_cpu_diag_59_rx.h	0b658136348769eae20302a02f55ed40
r_cpu_diag_30_rx.h	95a6de8506563d79f6b3de258acad9b6	r_cpu_diag_59_rx.src	c50ec8fd277a4d0975ea3af1e2ad716f
r_cpu_diag_30_rx.src	6e372b0152c1da5e4b53cb5d1efcb1f2	r_cpu_diag_60_rx.h	74a283769a42d94f123b6c70b82660fb
r_cpu_diag_31_rx.h	c2428d078c4f73b2ed0f46761a9353be	r_cpu_diag_60_rx.src	fb7368affaac24688ecf9a3428eb27f2
r_cpu_diag_31_rx.src	d6ac6a6c58b71ddaca09dbc044bf8b0d	r_cpu_diag_61_rx.h	6796e898b74ba4fc15e7001e0cd5561b
r_cpu_diag_32_rx.h	da02e78a6a335ea1565830adb692bedd	r_cpu_diag_61_rx.src	d5b12b194c220ccff0af6d7b54c1d9c0
r_cpu_diag_32_rx.src	cc254d38e8487a87cb31f3e6503546a8	r_cpu_diag_62_rx.h	cf6265545018a56dc0bef44b3fb4261a
r_cpu_diag_33_rx.h	01ebc77631432799c2d5053170cdd3c9	r_cpu_diag_62_rx.src	56c565e5c4f289709666dd10f36eb654
r_cpu_diag_33_rx.src	dd0005846a005a20bb32df4968252154	r_cpu_diag_63_rx.h	654485b0f6eea8d4f1e7b6cd3e825d97
r_cpu_diag_34_rx.h	f5e19f877b8a73e47f5e046bfdca1c26	r_cpu_diag_63_rx.src	0f7a91cf92723a3ca3b6b87aebeb4505
r_cpu_diag_34_rx.src	655266708abafed5e7b807511d5a87d0	r_cpu_diag_64_rx.h	fc361a6d370dde5190cb974fde5a5114
r_cpu_diag_35_rx.h	629933d6fa1f5a1e9df1daa70f45aa95	r_cpu_diag_64_rx.src	7b55523fc9a8b1c7d0a0301ec266ce53
r_cpu_diag_35_rx.src	821812addd681da823992d19dab8e2b0	r_cpu_diag_65_rx.h	286a8a6af8d00ff01b7cf76a2e9614e8
r_cpu_diag_36_rx.h	a13419804fe64d3bfe7d96b9e499c78f	r_cpu_diag_65_rx.src	a27bc12e6af310a17536da32ad0ed151
r_cpu_diag_36_rx.src	15a81292c688736b8a9bb677d30577c8	r_cpu_diag_66_rx.h	d5eea1f6d267f9475326e8bebca79556
r_cpu_diag_37_rx.h	15dbd509d038922b788d0f11433e3f2b	r_cpu_diag_66_rx.src	358c8b69616f97fd3d168f531de29a5c
r_cpu_diag_37_rx.src	635f03a6162a69ab6a0f100b1f00364e	r_cpu_diag_67_rx.h	6ad41b8710368ba5f335b61e63cb2600
r_cpu_diag_38_rx.h	14b07a5ae367fc300fd4009b2b3cc998	r_cpu_diag_67_rx.src	e0fc75a5496fac1315e18073112b4aa9
r_cpu_diag_38_rx.src	83e26ee8a2ff6e8d929e907b4bd05561	r_cpu_diag_68_rx.h	f37d77d0448aab133501127ff5444e22
r_cpu_diag_39_rx.h	1b4ed52b8d8940014c352e5f8aef04ad	r_cpu_diag_68_rx.src	325c3af9e881dc5ebbca331f53b57df4
r_cpu_diag_39_rx.src	27e45eb9206bfd70f026596d5145ab90	r_cpu_diag_69_rx.h	a62a30498d2ea1aed441f6cd6a114a22
r_cpu_diag_40_rx.h	52f56350aac5f04ace19cc027913c274	r_cpu_diag_69_rx.src	cbbd8b248de70e6a27032181cc85c52d
r_cpu_diag_40_rx.src	9b3eecda4f7df5568cc7fd84a9603027	r_cpu_diag_70_rx.h	d0210e0bd146d6b100bb8262a77c889e
r_cpu_diag_41_rx.h	e0d867ce80a7175cb0ced16b65924924	r_cpu_diag_70_rx.src	ffe082e7be3ff7de3bcab1312d92a718
r_cpu_diag_41_rx.src	2b36c2ea2b11d52066ca2483151c8ef0	r_cpu_diag_71_rx.h	b0e18819b169fb09b97192f861fdf8cb
r_cpu_diag_50_rx.h	2f643da658fdeaaf9640b2b88e2b3492	r_cpu_diag_71_rx.src	82a05a050beb9e18972e48691b0f032a
r_cpu_diag_50_rx.src	a24ff6146b138cd54c9f625b7030b985	r_cpu_diag_72_rx.h	1e43c755c9c30d842a1a2157138c73d9
r_cpu_diag_51_rx.h	2061b31f6a28e8edf3e4f2d40711d11f	r_cpu_diag_72_rx.src	6c219938ccfd5b0418537374a95ea596
r_cpu_diag_51_rx.src	e328a49aa5a6f44f4e93c76a5a3a652c	r_cpu_diag_73_rx.h	87fa2259ed1c23a9f8b9a81a50018da2
r_cpu_diag_52_rx.h	36bf096b90cc8eb1b1b2f0e3912f8763	r_cpu_diag_73_rx.src	21b87ddd87f5f4d4690ff8c0f84e5afd
r_cpu_diag_52_rx.src	3b941be411aced9a8714c2e013c39a35	r_cpu_diag_100_rx.h	9f28d3383c72beee7a239db9cd109fea
r_cpu_diag_53_rx.h	1b4f81790a9bd6e6268ca71f3fa2725b	r_cpu_diag_100_rx.src	dfd42894cefb64b415d29dbe74d2fc1f
r_cpu_diag_53_rx.src	84354332271aba4c7e52d1c48821d56d	r_cpu_diag_101_rx.h	b24f0ca2f4ee1c4c115bf321b41948c9
r_cpu_diag_54_rx.h	7808f971a8deabcb558619b7a70e81db	r_cpu_diag_101_rx.src	706341392328b54b6129017c8ccf4d6c
r_cpu_diag_54_rx.src	737bc4f847f16e1eb952bf038ebfa120	r_cpu_diag_102_rx.h	1e91118a4e538331a87af91811b6ea08
r_cpu_diag_55_rx.h	e431a47fbc8c23eb1692e37e4dad7446	r_cpu_diag_102_rx.src	4417cc9f7468c874428f91ef95358614
r_cpu_diag_55_rx.src	fb4d4b28be8af9159c1c0601fd4a0ca5	r_cpu_diag_103_rx.h	67ff2bea5a97aa8fb547024dc43cd4cb
r_cpu_diag_56_rx.h	a213a3ec949ec89d3190dd8b4e334468	r_cpu_diag_103_rx.src	51902c2dcdf5152a807357b6dfeedb3e
r_cpu_diag_56_rx.src	3a3a4a6e86f4946872a5ec9855efb87f	r_cpu_diag_104_rx.h	283a0679ec79587d22125e9ba34dbfa2
r_cpu_diag_57_rx.h	c8376fee9c106f4173f8cca08bb8ff0e	r_cpu_diag_104_rx.src	14285164c0fb70568f41d570185a7d98
r_cpu_diag_57_rx.src	0242bc752956c9ff76e4a79eb16e058c		

6.5.2 Identification and Contents of Package of RAM Test

Internal RAM test package version is identified as follows:

- Version 3.00

Table 6.13 Internal RAM test package and MD5 checksum

File name	MD5 checksum
testRAM.c	bc9d953d8576004879f344c230b2a892
testRAM.h	30a5a52f5d894b64a70fb17b4c5959f7
testRAM.inc	eee055589f0cf2ad897a2c5081af18f4
testRAM_config.h	101489ace56900263cee56f5c9894da2
extendedMarchCminus.h	6d3050983346764669 87c4e7053d9ad0
extendedMarchCminus.src	40a0dab75eab61addb0a389ddb6d1eaf
extendedMarchCminus_forRambuffer.h	bea735ed1e104ffcd70fb0bdab550315
extendedMarchCminus_forRambuffer.src	0aeee6969e9748adf2c2b2a022310589
walpat.h	80ad32a8949aaa5e29ddacbfb8aaebc1
walpat.src	eb939edf2efa4925cb7f773c262b5ac6
walpat_forRambuffer.h	f662035dc1001e492fe54c56461acf44
walpat_forRambuffer.src	8409c83071a411c66556ddbeb7f0a4fd

6.5.3 Identification and Contents of Package of ROM Test

Internal ROM test package version is identified as follows:

- Version 3.00

Table 6.14 Internal ROM test package and MD5 checksum

File name	MD5 checksum
crc.src	88584c6a12dbd9c715505831127416c2
testROM.h	2ec7dfa54c4cd80ee4b59f466bc24f29
testROM_config.inc	e2902b1a9065edaa444a5cdf4d3fe9ac

7. Revision History

Revision History	RX(v2/v3 core) Family Diagnostic Software : User Guide

Rev.	Date	Description	
		Page	Summary
1.00	Jul 27, 2018	–	First edition (supporting V2 core)
2.00	Feb 14, 2020	–	Code optimization. Adaptation to V3 core.
3.00	Sep 30, 2020	–	Integrated RX700/600/200 series into one.

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

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

1. Precaution against Electrostatic Discharge (ESD)

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

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

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

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

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

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

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

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

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.
   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user's manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION ("Vulnerability Issues"). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

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

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

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.