Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

The following document contains information on Cypress products. The document has the series name, product name, and ordering part numbering with the prefix “MB”. However, Cypress will offer these products to new and existing customers with the series name, product name, and ordering part number with the prefix “CY”.

How to Check the Ordering Part Number

1. Go to www.cypress.com/pcn.
2. Enter the keyword (for example, ordering part number) in the **SEARCH PCNS** field and click **Apply**.
3. Click the corresponding title from the search results.
4. Download the Affected Parts List file, which has details of all changes

For More Information

Please contact your local sales office for additional information about Cypress products and solutions.

About Cypress

Cypress is the leader in advanced embedded system solutions for the world's most innovative automotive, industrial, smart home appliances, consumer electronics and medical products. Cypress' microcontrollers, analog ICs, wireless and USB-based connectivity solutions and reliable, high-performance memories help engineers design differentiated products and get them to market first. Cypress is committed to providing customers with the best support and development resources on the planet enabling them to disrupt markets by creating new product categories in record time. To learn more, go to www.cypress.com.

AN204377

FM3 and FM4 Family, IEC61508 SIL2 Self-Test Library

Associated Part Family: Refer to section 2

This Application Note addresses those of you who consider using the Self-Test Library (STL) for MCU by means of software supporting an equivalent of SIL2 for the IEC61508 in the FM3 family and FM4 family of MCU manufactured by Cypress.

Contents

1 Introduction

1.1 About This Application Note

This Application Note addresses those of you who consider using the Self-Test Library (STL) for MCU by means of software supporting an equivalent of SIL2 for the IEC61508 in the FM3 family and FM4 family of MCU manufactured by Cypress.

Note: The Self Test Library is available upon request. To request the library, please contact your local distributor.

1.2 Background

Amid growing importance being attached to the concept and methodology of keeping systems safe in various industrial sectors, there are underway globally to pursue standardization. IEC61508, formulated by IEC (International Electrotechnical Commission), covers an extensive range of industrial sectors in which any other safety standard is not yet in place. The formulation of IEC61508 has resulted in increased urges for standard compliance in product development activities.

1.3 Development and Evaluation Environment

This STL has been developed and evaluated in the environment described in Table 1

Table 1. Development and Evaluation Environment Table

Microprocessor	[FM3] MB9BF506R [FM4] MB9BF568R
IDE	IAR Embedded Workbench for ARM6.60 kickstart KEIL µVision V5.0.5.15
Evaluation board	[FM3(80MHz,Flash access 2-wait)] MB9BF506R-SK MCB9B500 [FM4(160MHz,Flash Accelerator Enable)] SK-FM4-U120-9B560
Optimization	[IAR] high (balanced) [KEIL]Level3

1.4 Note

Since the STL provided has not been certified by a certification authority, using it without modification does not necessarily ensure certification. This STL is designed on the assumption that SIL2 is fulfilled with an SFF of 90% or more but less than 99%. (See "3.5 Hardware Safety Evaluations and Architectural Constraints".)

2 Target Products

This application note is described about below products;

(FM3 :TYPE0)

Series	Product Number (not included Package suffix)
MB9B500B	MB9BF504NB,MB9BF505NB,MB9BF506NB MB9BF504RB,MB9BF505RB,MB9BF506RB
MB9B400A	MB9BF404NA,MB9BF405NA,MB9BF406NA MB9BF404RA,MB9BF405RA,MB9BF406RA
MB9B300B	MB9BF304NB,MB9BF305NB,MB9BF306NB MB9BF304RB,MB9BF305RB,MB9BF306RB
MB9B100A	MB9BF102NA,MB9BF104NA,MB9BF105NA,MB9BF106NA MB9BF102RA,MB9BF104RA,MB9BF105RA,MB9BF106RA

(FM4)

Series	Product Number (not included Package suffix)
MB9B560R	MB9BF566R,MB9BF567R,MB9BF568R MB9BF566N,MB9BF567N,MB9BF568N MB9BF566M,MB9BF567M,MB9BF568M
MB9B460R	MB9BF466R,MB9BF467R,MB9BF468R MB9BF466N,MB9BF467N,MB9BF468N MB9BF466M,MB9BF467M,MB9BF468M

Series	Product Number (not included Package suffix)
MB9B360R	MB9BF366R,MB9BF367R,MB9BF368R MB9BF366N,MB9BF367N,MB9BF368N MB9BF366M,MB9BF367M,MB9BF368M
MB9B160R	MB9BF166R,MB9BF167R,MB9BF168R MB9BF166N,MB9BF167N,MB9BF168N MB9BF166M,MB9BF167M,MB9BF168M

3 Overview of IEC61508

3.1 Glossary

Table 2 lists the terms used in this Application Note and their definitions.

Table 2. Terms/Definitions

Term	Definition
Safety-related system	A system designed to reduce the risk potentials that threaten a particular control system to a tolerable level or below to maintain safety.
Safety function	The function of a safety-related system
Functional safety	To reduce risk potentials to a tolerable level or below by means of safety function to maintain safety.
Safety lifecycle	A 16-phase sequence of working processes in which requirements for a safety-related systems are defined, ranging from conceptualization to decommissioning or disposal.
SIL	Short for Safety Integrity Level, or the level of risks with which a safety-related system fails to implement its safety function. There are four SILs, from 1 to 4, 4 being the highest.
Proof test	A period test that is carried out to detect failures in a safety-related system that cannot be detected automatically.
Low demand mode of operation	The mode of operation of a safety-related system in which requests for its operation occur once (each year) or less often, or less often than twice the proof test frequency.
High demand mode of operation	The mode of operation of a safety-related system in which requests for its operation occur once (each year) or more often, or more often than twice the proof test frequency.
Continuous mode	The mode of operation of a safety-related system in which its safety function is at constant work as part of its normal operation.
PFDavg	Short for Average Probability of dangerous Failure on Demand, or the average probability of dangerous failures on demand for the activation of safety function.
PFH	Short for Probability of dangerous Failure per Hour, or the average probability of dangerous failures per unit hour.
Random failure	A failure that occurs at random timings, such as a hardware fault.
Systematic failure	A failure, such as a software bug, that necessarily occurs as a result of its cause involved in a development process.
Safe failure rate	The failure of a safety-related system that does not risk a dangerous situation for the controlled object.
Dangerous failure rate	The failure of a safety-related system that risks a dangerous situation for the controlled object.
DC	Short for Diagnostic Coverage, or the ratio of the number of detectable dangerous failures to the total number of dangerous failures.
SFF	Short for Safe Failure Fraction, or the ratio of the total number of safe failures and detectable dangerous failures to the total number of failures.
Fault tolerance	The ability of a functional unit to implement its safety function despite the occurrence of faults.

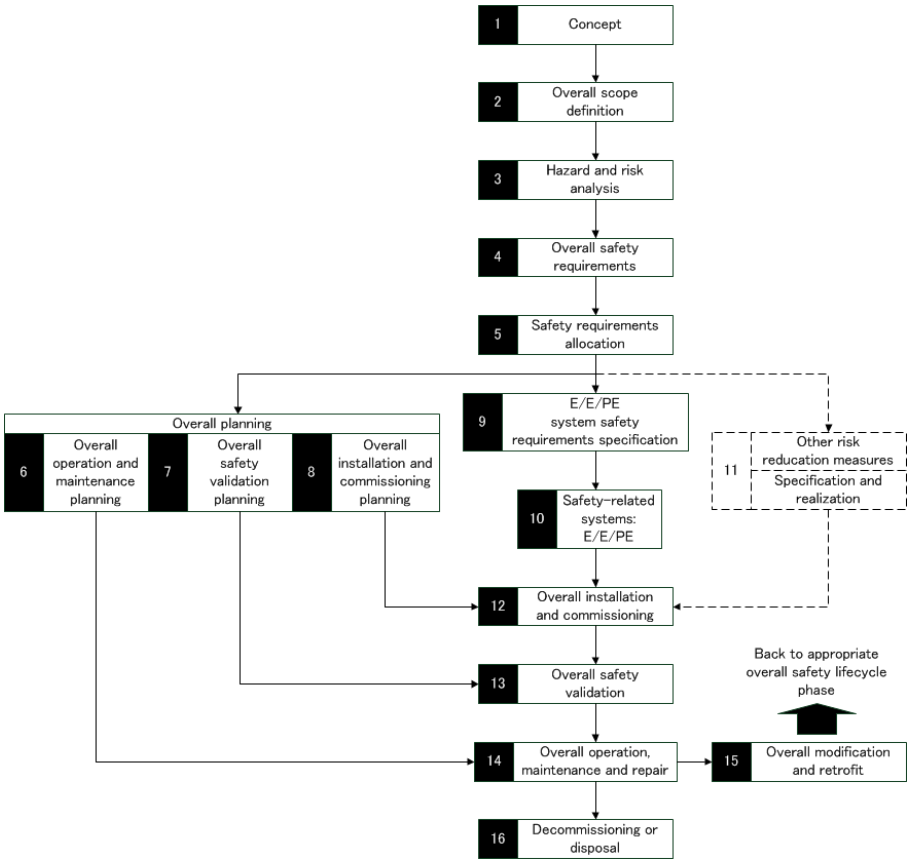
Term	Definition
Type	Any safety-related system that meets all of the framework conditions listed below is called Type A, and all other systems are called Type B. Type A is typified by resistors and capacitors, whereas Type B is represented by ASICs. Microprocessors are considered to fall in Type B. Capable of defining the failure modes of all its components; Capable of determining the behavior of its subsystems completely through failure conditioning, and Holds fully dependable field data that supports the claims of detected and yet-to-be-detected dangerous failure rates.

3.2 Safety lifecycle

A system that is designed to keep a particular control system safe is called a "safety-related system." Requirements for safety-related systems (which are composed solely of electrical/electronic/programmable electronic systems (E/E/PES) are defined in IEC61508.

Defined requirements for a safety-related system range from conceptualization, through design, development and maintenance, to decommissioning or disposal. This sequence of processes is called a "safety lifecycle." (See Figure 1) All activities that are directed at a safety-related system are expected to be administered in this safety life cycle.

Figure 1. Safety Lifecycle



3.3 SIL (Safety Integrity Level)

IEC61508 has tolerances established for the frequency of failure occurrence to apply to the safety function of a safety-related system. Indicators of these tolerances are called "Safety Integrity Levels (SILs)". There are four SILs, from 1 to 4. A specific numeric goal is defined for each SIL as listed in Table 3.

Table 3. Target Failure Measures of Safety Function

SIL	Low demand mode of operation (PFDavg)	High demand mode of operation or continuous mode (PFH)
4	$\geqq 10^{-5}$ to $< 10^{-4}$	$\geqq 10^{-9}$ to $< 10^{-8}$
3	$\geqq 10^{-4}$ to $< 10^{-3}$	$\geqq 10^{-8}$ to $< 10^{-7}$
2	$\geqq 10^{-3}$ to $< 10^{-2}$	$\geqq 10^{-7}$ to $< 10^{-6}$
1	$\geqq 10^{-2}$ to $< 10^{-1}$	$\geqq 10^{-6}$ to $< 10^{-5}$

The SIL is established through phases 1 to 5 of the safety lifecycle. The subsequent phases proceed to meet the SIL thus established.

3.4 Failure Definitions

IEC61508 classifies failures occurring in a safety-related system into random failures and systematic failures. Assuming that the safe failure rate is λ_S, the dangerous failure rate is λ_D and the total failure rate is λ, equation (1) holds true.

Equation 1. $\lambda = \lambda S + \lambda D$

The failures can be further classified into detectable and non-detectable failures as summarized in Table 4.

Table 4. Failure Classifications

	Detectable	Non-detectable
Safe	Safe failures that allow self-testing	Safe failures that resist self-testing
Dangerous	Dangerous failures that allow self-testing (λ_{DD})	Dangerous failures that reject self-testing (λ_{DU})

Equation (2) is derived from Table 4.

Equation 2. $\lambda_D = \lambda_{DD} + \lambda_{DU}$

DC where is the ratio of the number of failures that can be detected by self-testing can be expressed in an equation (3) as

Equation 3. $DC = \lambda_{DD} / \lambda_D$

The value of DC weighs in considering the SFF (safe failure fraction). (For more information about the SFF, see 3.6.)

3.5 Hardware Safety Evaluations and Architectural Constraints

It's expected that systematic failures should be checked by running the safety lifecycle, and random failures should be checked by implementing a safe hardware architectural design that fulfills safety requirements, respectively.

The process of evaluating hardware safety should not only fulfill the target failure measures of safety function listed in Table 3 but also achieve a certain level of fault tolerance in the subsystems that make up the safety-related system. An upper limit to the SIL that can be claimed is determined by the three factors of such hardware fault tolerance, subsystem type and SFF. This is called an "architectural constraint." Type A and Type B architectural constraints are listed in Table 5 and Table 6, respectively. N means that a loss of the safety function might be incurred by the occurrence of N+1 faults.

Table 5. Type A Architectural Constraints

SFF	Hardware fault tolerance (N)		
	0	1	2
< 60%	SIL1	SIL2	SIL3
60% - < 90%	SIL2	SIL3	SIL4
90% - < 99%	SIL3	SIL4	SIL4
≧ 99%	SIL3	SIL4	SIL4

Table 6. Type B Architectural Constraints

SFF	Hardware fault tolerance (N)		
	0	1	2
< 60%	Not allowed	SIL1	SIL2
60% - < 90%	SIL1	SIL2	SIL3
∗90% - < 99%	SIL2	SIL3	SIL4
≧ 99%	SIL3	SIL4	SIL4

∗ The target of our STL.

3.6 SFF (Safe Failure Fraction)

Under IEC61508, the SFF is defined in Equation (4) as

Equation 4. $SFF = (\Sigma\lambda_S + \Sigma\lambda_{DD}) / (\Sigma\lambda_S + \Sigma\lambda_D))$

This means that the SFF is the ratio of the fraction of safe failures to that of all failures. λ_{DD} and λ_S are to be handled in an equivalent manner on the assumption that self-testing is fully effective.

Equation (5) represents the relationship between the SFF and DC on the basis of Equations (2) and (4), where $\eta = \Sigma\lambda_S / \Sigma\lambda_D$ is assumed.

Equation 5. $SFF = (\eta + DC) / (\eta + 1)$

η=1 is assumed for semiconductor devices in most situations. Hence, Equation (6) is derived.

Equation 6. $SFF = 0.5 + 0.5*DC$

The value of DC is so important, such that the SFF depends on it. For this reason, the components that should be heeded in calculating the SFF and the associated test methods are defined in IEC61508-2 Annex A, along with the maximum values of DC available during use of the test methods.

4 Overview of STL

4.1 Coverage of Testing

Table 7 lists those types of components under test that are defined in IEC61508-2 Annex A and the associated test methods, that are supported by our STL.

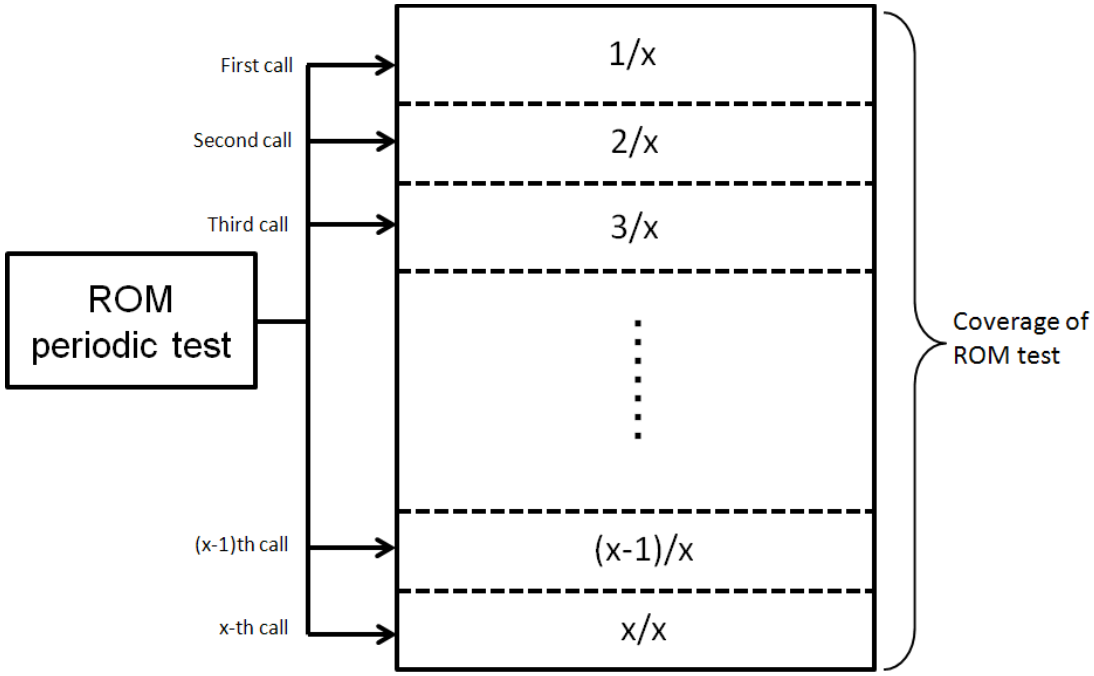
Table 7. Coverage of Testing

Component	Coverage	Test method	Maximum value of DC
BUS	Data paths	Inspection using test patterns	99%≦
CPU	Processing units	Self-test by software: walking bit (one-channel)	90%≦ to <99%
ROM	Invariable memory ranges	Signature of a double word (16-bit)	99%≦
RAM	Variable memory ranges	RAM test galpat or transparent galpat	99%≦

4.2 Types of Testing

Two types of testing are supported for each component: startup test and periodic test. The startup test is to be carried out at reset time. Once called, the startup test tests the entire range of a component in question. The periodic test, on the other hand, is to be performed at intervals of the main loop. It tests a component in question in installments. It is run several times to test the entire range of the component. Where the periodic ROM test is divided into x segments, for example, if the test is called x times, then the result would be as shown in Figure 2.

Figure 2. ROM Periodic Call (x segments)



4.3 Failure Detection

Each time a failure is detected in a session of testing, it is counted and the same session of testing is continued further. Users can set a threshold for failure detection. When a count of failures detected exceeds this threshold, the failure state is turned on to report the occurrence of a failure in the component in question.

4.4 STL Configuration

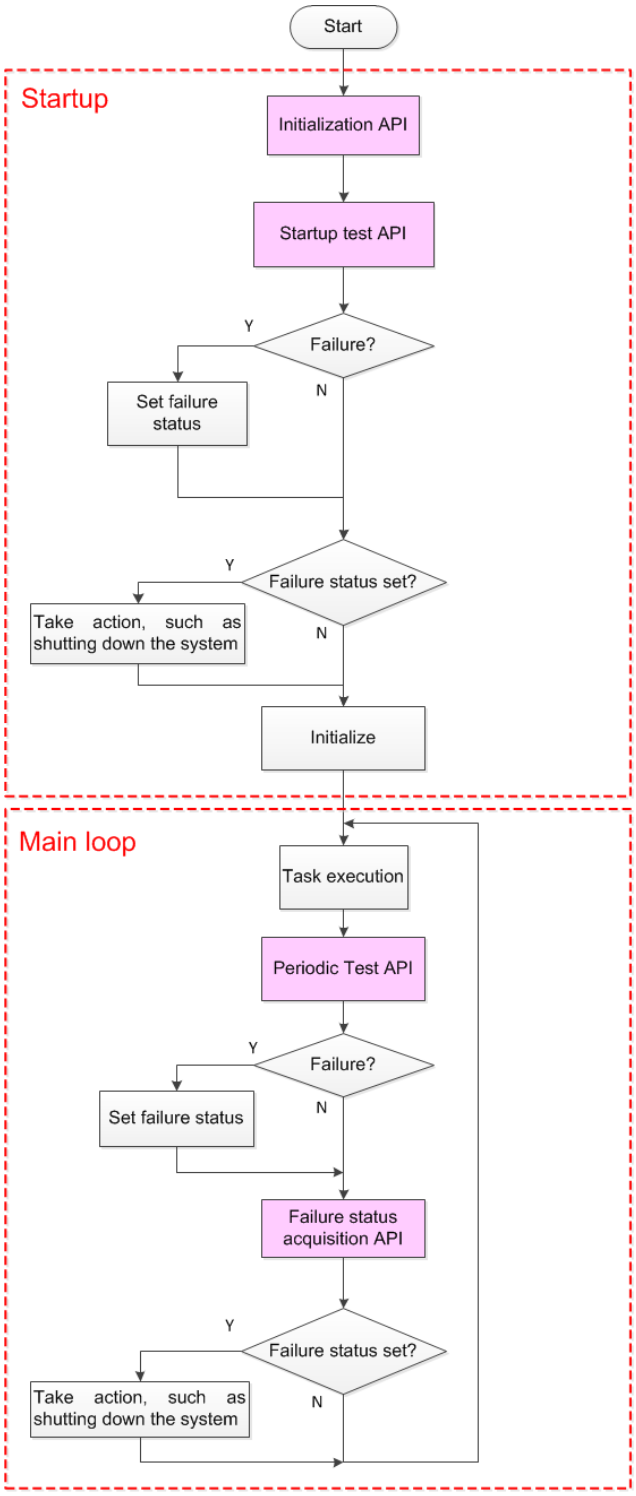
The STL maintains the APIs listed in Table 8 for each component.

Table 8. API Types

API	Overview
Initialization API	Initializes the variables used during testing.
Startup test API	Runs a startup test.
Periodic test API	Runs a periodic test.
Failure state acquisition API	Gets the latest failure state.

A sample run of APIs by the STL is flowcharted in Figure 3. This sample run is common to all components.

Figure 3. API Sample Run



5 Overview of API

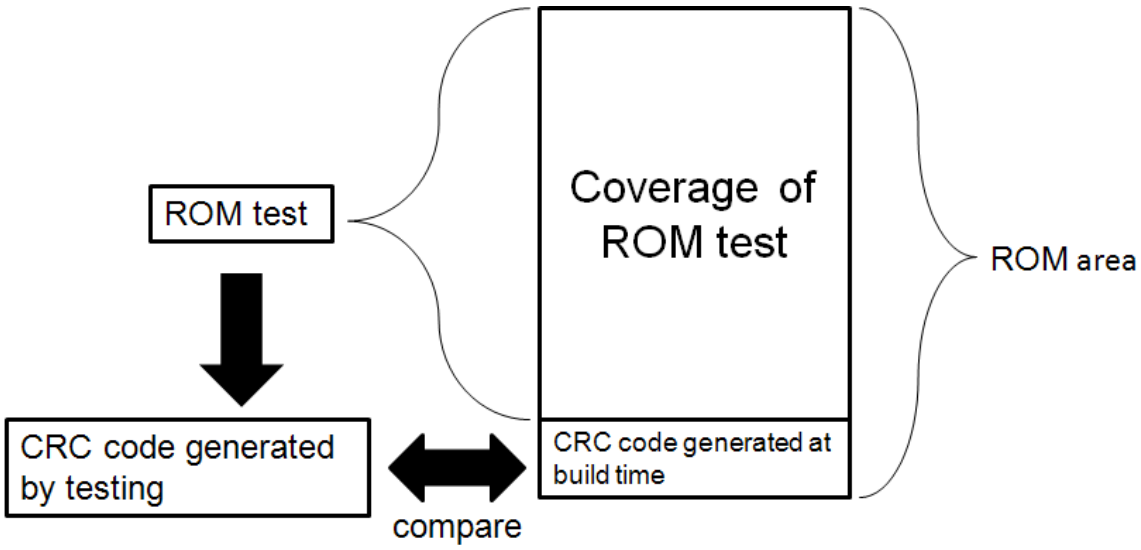
5.1 ROM Test

5.1.1 Test Overview

Test method ”Signature of a double word (16-bit)” is used to test ROM. For more information, refer to IEC61508-7 A4.4.

An overview of the ROM test is shown in Figure 4.The CRC code that has been generated during at build time is placed in a particular ROM area. Then, a CRC code is generated by running a startup or periodic test after the system has started up. This CRC code is compared with the CRC code generated at build time to check that the two versions of the CRC code match. This STL uses CRC32 offering higher detection accuracy than CRC16 to compute the CRC code. The CRC code that is generated by testing may be computed either by software or by using a hardware macro at the users' option.

Figure 4. ROM Test Schematic



5.1.2 Programmable Parameters

User-programmable parameters in using the ROM test API are as follows:

1. Number of areas under test
2. Start and end addresses of each area under test
3. Threshold for failure detection
4. Test length calculated in a single run of the periodic test (bytes)

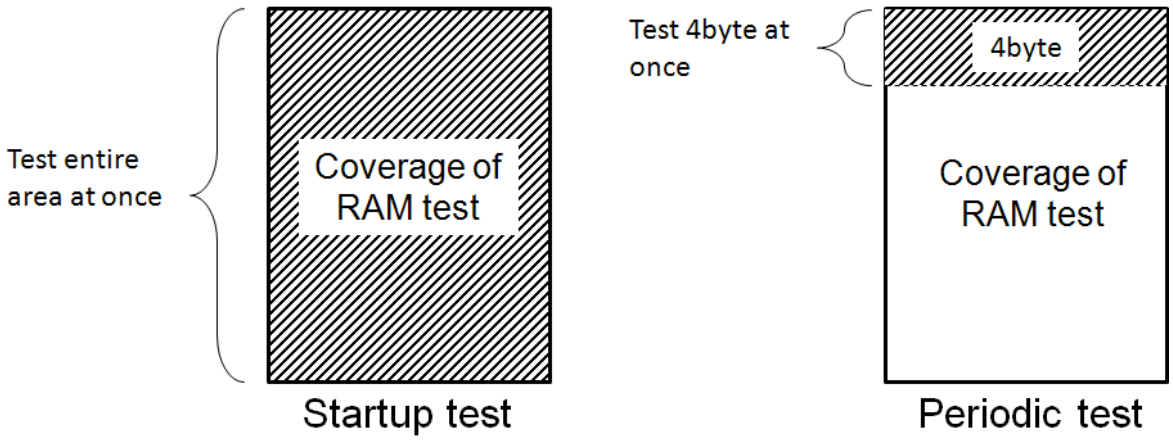
5.2 RAM Test

5.2.1 Test Overview

Test method ”RAM test galpat or transparent galpat” is used to test RAM. For more information, refer to IEC61508-7 A5.3.

Both startup test and periodic test use galpat. A startup test is performed across an area of interest, whereas a periodic test is run for a segment of 4 bytes at a time. (See Figure 5).

Figure 5. RAM Test



5.2.2 Programmable Parameters

User-programmable parameters in using the RAM test API are as follows:

1. Number of areas under test
2. Start and end addresses of each area under test (which may be differentiated between the startup and periodic tests)
3. Threshold for failure detection

5.3 CPU Test

5.3.1 Test Overview

Test method “Self-test by software: walking bit (one-channel)” is used to test CPUs. For more information, refer to IEC61508-7 A3.2.

The CPU test tests for the normal operations of each of the instructions in the 16-bit ARMv7-M Thumb instruction set (FM3 family and FM4 family are common. When inspecting the Thumb-2 instruction set is necessary, separate preparation is required) listed in Table 9 and each of the registers (R0 to R14). Testing is carried out by computing a walking-bit pattern as an input value and checking to see if the result of the computation is as expected or not.

Table 9. ARMv7-M Thumb Instruction Set (16-bit)

No.	Instruction	Syntax	Description
1	ADC	<Rd>, <Rm>+	Add the register value and the C-flag to the register value. Rd = Rd + Rm + C
2	ADD	<Rd>, <Rn>, #<immed_3>+	Add a 3-bit immediate value to the register. Rd + Rn + immed_3
3	ADD	<Rd>, #<immed_8>+	Add an 8-bit immediate value to the register. Rd + Rd + immed_8

No.	Instruction	Syntax	Description
4	ADD	<Rd>, <Rn>, <Rm>+	Add the lower register value to the lower register value. Rd = Rn + Rm
5	ADD	<Rd>, <Rm>	Add the upper register value to the lower or upper register value. Rd = Rn + Rm
6	ADD	<Rd>, PC, #<immed_8>*4	Add the PC address + 4 × (8-bit immediate value) to the register. Rd = PC + 4*immed_8
7	ADD	<Rd>, SP, #<immed_8>*4	Add the SP address + 4 × (8-bit immediate value) to the register. Rd = SP + 4*immed_8
8	ADD	SP, #<immed_7>*4	Add 4 × (7-bit immediate value) to SP. SP = SP + 4*immed_7
9	AND	<Rd>, <Rm>+	AND the register value bitwise. Rd = Rd AND Rm
10	ASR	<Rd>, <Rm>, #<immed_5>+	Shift to right arithmetically depending on the immediate value. Rd = Rd >> Rs
11	B<cond>	<target_address_8>	Conditional branch if <cond> then PC = (PC + 4) + (SignExtend (target_address_8) * 2)
12	B	<target_address_11>	Unconditional branch PC = (PC + 4) + (SignExtend (target_address_11) * 2)
13	BIC	<Rd>, <Rm>+	Bit clear Rd = Rd AND (NOT Rm)
14	BKPT	<immed_8>	Software breakpoint
15	BL	<target_address11>	Linked branch
16	BLX	<Rm>	Linked branch and state transition (Rm[bit 0] set to 1)
17	BX	<Rm>	Branch and state transition (Rm[bit being 0] set to 1)
18	CBNZ	<Rn>, <label>	Branch if non-zero on comparison (forward branch only).
19	CBZ	<Rn>, <label>	Branch if 0 on comparison (forward branch only).
20	CMN	<Rn>, <Rm>+	Compare two's complement of the register value (negative, negation) with the value of another register (Rn - (-Rm)) and update the flag accordingly.
21	CMP	<Rn>, #<immed_8>+	Compare the register value with an 8-bit immediate value.
22	CMP	<Rn>, <Rm>+	Compare with the register.
23	CMP	<Rn>, <Rm>+	Compare the upper register with the lower or upper register.
24	CPSIE CPSID	< i or f > < i or f >	Processor status change CPSIE clears PRIMASK(i) or FAULTMASK(f) to enable interrupt. CPSID clears PRIMASK(i) or FAULTMASK(f) to disable interrupt.

No.	Instruction	Syntax	Description
25	CPY	<Rd>, <Rm>	Copy the upper register or lower register value to another upper or lower register.
26	EOR	<Rd>, <Rm>+	Exclusive OR the register value bitwise.
27	IT IT<x> IT<x><y> IT<x><y><z>	<cond> <cond> <cond> <cond>	Instruction to create a block with an IF-THEN condition. One to four instructions that follow this instruction can be executed conditionally on the basis of condition <cond>.
28	LDMIA	<Rn>!, <registers>	Load Multiple Increment After Load multiple words starting from the address specified by Rn into the register.
29	LDR	<Rd>, [<Rn>, #<immed_5>*4]	Load a word from the memory location addressed by the base register offset by a 5-bit immediate value.
30	LDR	<Rd>, [<Rn>, <Rm>]	Load a word from the memory location addressed by the base register offset by the register.
31	LDR	<Rd>, [PC, #<immed_8>*4]	Load a word from the memory location addressed by PC offset by an 8-bit immediate value.
32	LDR	<Rd>, [SP, #<immed_8>*4]	Load a word from the memory location addressed by SP offset by and 8-bit immediate value.
33	LDRB	<Rd>, [<Rn>, #<immed_5>]	Load byte [7:0] from the memory location addressed by the base register offset by a 5-bit immediate value into the register.
34	LDRB	<Rd>, [<Rn>, <Rm>]	Load byte [7:0] from the memory location addressed by the base register offset by the register into the register.
35	LDRH	<Rd>, [<Rn>, #<immed_5>*2]	Load halfword [15:0] from the memory location addressed by the base register offset by a 5-bit immediate value into the register.
36	LDRH	<Rd>, [<Rn>, <Rm>]	Load halfword [15:0] from the memory location addressed by the base register offset by the register into the register.
37	LDRSB	<Rd>, [<Rn>, <Rm>]	Load byte [7:0] from the memory location addressed by the base register offset by the register into the register, signed.
38	LDRSH	<Rd>, [<Rn>, <Rm>]	Load halfword [15:0] from the memory location addressed by the base register offset by the register into the register, signed.
39	LSL	<Rd>, <Rm>, #<immed_5>+	Shift left logically depending on the immediate value. Rd = Rd << immed_5
40	LSL	<Rd>, <Rs>+	Shift left logically depending on the register value. Rd = Rd << Rs
41	LSR	<Rd>, <Rm>, #<immed_5>+	Shift right logically depending on the immediate value. Rd = Rd >> immed_5
42	LSR	<Rd>, <Rs>+	Shift right logically depending on the register value. Rd = Rd >> Rs
43	MOV	<Rd>, #<immed_8>+	Move the 8-bit immediate value to the register.
44	MOV	<Rd>, <Rn>+	Move the lower register value to the lower register.

No.	Instruction	Syntax	Description
45	MOV	<Rd>, <Rm>	Move the upper or lower register value to the upper or lower register.
46	MUL	<Rd>, <Rm>+	Multiply the register value. Rd = Rd * Rm
47	MVN	<Rd>, <Rm>+	Move the negation of the register value (one's complement) to the register. Rd = NOT(Rm)
48	NEG	<Rd>, <Rm>+	Convert the register value to negative (two's complement, negative) and save to the register. Rd = 0 –Rm
49	NOP		No operation
50	ORR	<Rd>, <Rm>+	OR the register value bitwise. Rd = Rd OR Rm
51	POP	<registers>	Pop multiple registers from the stack.
52	POP	<registers, PC>	Pop multiple registers and PC from the stack.
53	PUSH	<registers>	Push multiple registers to the stack.
54	PUSH	<registers, LR>	Push multiple registers and LR to the stack.
55	REV	<Rd>, <Rn>	Copy a word to the register in inverted byte order. Rd = {Rn[7:0], Rn[15:8], Rn[23:16], Rn[31:24]}
56	REV16	<Rd>, <Rn>	Copy two halfwords to the register each in inverted byte order. Rd = {Rn[23:16], Rn[31:24], Rn[7:0], Rn[15:8]}
57	REVSH	<Rd>, <Rn>	Copy lower halfword [15:0] and copy it to the register in inverted byte order, sign-extended. Rd = SignExtend ({Rn[7:0], Rn[15:8]})
58	ROR	<Rd>, <Rs>+	Rotate right by the value specified by the register.
59	SBC	<Rd>, <Rm>+	Subtract the register value and a borrow (~C) from the register value. Rd = Rd - Rm -NOT(C)
60	SEV		
61	STMIA	<Rn>!, <registers>	Store multiple registers to a sequence of memory locations in words.
62	STR	<Rd>, [<Rn>, #<immed_5>*4]	Store the register value at the memory location addressed by the register offset by a 5-bit immediate value as a word.
63	STR	<Rd>, [<Rn>, <Rm>]	Store the register value at the memory location addressed by the base register offset by the register as a word.
64	STR	<Rd>, [SP, #<immed_8>*4]	Store the register value at the memory location addressed by SP offset by an 8-bit immediate value as a word.
65	STRB	<Rd>, [<Rn>, #<immed_5>]	Store the register value at the memory location addressed by the register offset by a 5-bit immediate value as a byte [7:0].
66	STRB	<Rd>, [<Rn>, <Rm>]	Store the register value at the memory location addressed by the register offset as byte [7:0].

No.	Instruction	Syntax	Description
67	STRH	<Rd>, [<Rn>, #<immed_5>*2]	Store the register value at the memory location addressed by the register offset by a 5-bit immediate as halfword [15:0].
68	STRH	<Rd>, [<Rn>, <Rm>]	Store the register value at the memory location addressed by the register offset as halfword [15:0].
69	SUB	<Rd>, <Rn>, #<immed_3>+	Subtract a 3-bit immediate value from the register. Rd = Rn -immed_3
70	SUB	<Rd>, #<immed_8>+	Subtract an 8-bit immediate value from the register. Rd = Rd -immed_8
71	SUB	<Rd>, <Rn>, <Rm>+	Decrement the register value. Rd = Rn - Rm
72	SUB	SP, #<immed_7>*4	Subtract 4 × (7-bit immediate value) from SP. SP = SP - immed_7*4
73	SVC	<immed_8>	Call the operating system services (supervisor call) with an 8-bit immediate value call code.
74	SXTB	<Rd>, <Rm>	Extract byte [7:0] from the register and move it to the register to sign-extend to 32 bits.
75	SXTH	<Rd>, <Rm>	Extract halfword [15:0] from the register and move it to the register to sign-extend to 32 bits.
76	TST	<Rn>, <Rm>+	AND with another register value to test the set bits of the register. Rn AND Rm
77	UXTB	<Rd>, <Rm>+	Extract byte [7:0] from the register and move it to the register to zero-extend to 32 bits.
78	UXTH	<Rd>, <Rm>+	Extract halfword [15:0] from the register and move it to the register to zero-extend to 32 bits.
79	WFE		Wait for an event.
80	WFI		Wait for an interrupt.

Nos. 49, 60, 79 and 80 are excluded.

FM4 family, to test the instruction of ARMv7-M Thumb instruction set other than. From the instruction set of Cortex-M4 supports, to perform the DSP multiply instruction, the test of the FPU multiply instruction. (If other tests instruction is needed, will require additional) Inside the CPU core, it is not possible to judge whether all paths from the register to the DSP and FPU in all instructions are identical. Therefore, change the register in each instruction for testing. At that time, use the walking bit pattern for the test input value. FPU multiply instruction and DSP multiply instruction in the following list is tested.

Table 10. ARMv7-M Instruction Set (DSP)

No.	Instruction	Syntax	Description
1	SMMLA	SMMLA <Rd>,<Rn>,<Rm>,<Ra>	Signed Most Significant Word Multiply Accumulate multiplies two signed 32-bit values, extracts the most significant 32 bits of the result, and adds an accumulate value.
2	SMMLS	SMMLS <Rd>,<Rn>,<Rm>,<Ra>	Signed Most Significant Word Multiply Subtract multiplies two signed 32-bit values, subtracts the result from a 32-bit accumulate value that is shifted left by 32 bits, and extracts the most significant 32 bits of the result of that subtraction.
3	SMMUL	SMMUL <Rd>,<Rn>,<Rm>	Signed Most Significant Word Multiply multiplies two signed 32-bit values, extracts the most significant 32 bits of the result, and writes those bits to the destination register.
4	SMMLAR	SMMLAR <Rd>,<Rn>,<Rm>,<Ra>	Signed Most Significant Word Multiply Accumulate multiplies two signed 32-bit values, extracts the most significant 32 bits of the result, and adds an accumulate value.
5	SMMLSR	SMMLSR <Rd>,<Rn>,<Rm>,<Ra>	Signed Most Significant Word Multiply Subtract multiplies two signed 32-bit values, subtracts the result from a 32-bit accumulate value that is shifted left by 32 bits, and extracts the most significant 32 bits of the result of that subtraction.
6	SMMULR	SMMULR <Rd>,<Rn>,<Rm>	Signed Most Significant Word Multiply multiplies two signed 32-bit values, extracts the most significant 32 bits of the result, and writes those bits to the destination register.

Table 11. ARMv7-M Instruction Set (FPU)

No.	Instruction	Syntax	Description
1	VMUL.F32	VMUL.F32 <Sd>, <Sn>, <Sm>	Floating-point Multiply multiplies two single-precision register values, and places the result in the destination single-precision register.
2	VMLA.F32	VMLA.F32 <Sd>, <Sn>, <Sm>	Floating-point Multiply Accumulate multiplies two single-precision registers, and accumulates the results into the destination single-precision register.
3	VMLS.F32	VMLS.F32 <Sd>, <Sn>, <Sm>	Floating-point Multiply Accumulate multiplies two single-precision registers, and accumulates the results into the destination single-precision register.
4	VNMLA.F32	VNMLA.F32 <Sd>, <Sn>, <Sm>	Floating-point Multiply Accumulate and Negate multiplies two single-precision register values, adds the negation of the single-precision value in the destination register to the negation of the product, and writes the result back to the destination register.
5	VNMLS.F32	VNMLS.F32 <Sd>, <Sn>, <Sm>	Floating-point Multiply Accumulate and Negate multiplies two single-precision register values, adds the negation of the single-precision value in the destination register to the negation of the product, and writes the result back to the destination register.

5.3.2 Programmable Parameters

User-programmable parameters in using the CPU test API are as follows:

1. Test instructions
2. Threshold for failure detection

5.4 Bus Test

5.4.1 Test Overview

Test method “Inspection using test pattern” is to test buses. For more information, refer to IEC61508-7 A7.4.

The functions that are tested by the bus test are listed below.

[Bus functions under test]

- I-code bus
- D-code bus
- System bus
- DMAC bus
- DSTC bus (FM4)
- Bit-banding
- Unaligned access
- Arbitration

The bus periodic test is carried out by dividing the bus functions under test into three groups (FM3), or into four groups (FM4).

5.4.2 Programmable Parameters

The user-programmable parameter in using the bus test API is as follows:

1. Threshold for failure detection

6 STL Performance

The following table shows the performance of the STL with "ARM6.60 kickstart version of IAR Embedded Workbench for". It is a measurement result of FM3 (MB9BF506R).

6.1 ROM/RAM Sizes

Table 12 lists the sizes of ROM and RAM by component.

Table 12. ROM/RAM Sizes

(in bytes)

	ROM size (total)	RAM size (total)
RAM test	584	24
ROM test*	298	28
Bus test	784	248
CPU test	7742	148
Total	9408	448

*: The ROM size that is cited for the ROM test applies where a hardware macro is used to calculate the CRC code. If the software is used, the ROM size would be 1,306 (bytes).

6.2 Test Periods

This section presents the test periods by component. The indicated periods have been measured on a normal.

6.2.1 RAM Test Periods

Table 13 and Table 14 list the RAM startup test and periodic test periods, respectively.

Table 13. RAM Startup Test Periods

Number of areas	Area under test (byte)	Startup test (ms)
1	100	12
1	500	278
1	1000	1100
8	100	14
8	500	290
8	1000	1130

Table 14. RAM Periodic Test Periods

Area under test (byte)	Periodic test(μs)
4 (fixed)	19.6

6.2.2 ROM Test Periods

Table 15 and Table 16 list the ROM startup test and periodic test periods, respectively.

Table 15. ROM Startup Test Periods

Method of computing the CRC code	Number of areas	Area under test (byte)	Startup test (μs)
HW	1	500	110
HW	1	1000	211
HW	4	500	110
HW	4	1000	211
SW	1	500	380
SW	1	1000	752
SW	4	500	380
SW	4	1000	752

Table 16. ROM Periodic Test Periods

Method of computing the CRC code	Area under test (byte)	Periodic test (μs)
HW	10	9.6
HW	50	13.8
HW	100	19
SW	10	4.5
SW	50	19.4
SW	100	37.6

6.2.3 CPU Test period

The CPU startup test takes 246µs to execute.

Table 17 lists the CPU periodic test periods. Measurement is carried for each instruction set (for example, LDR set for LDR, LDRB and LDRH).

Table 17. CPU Periodic Test Periods

Instruction set	Periodic test (µs)
ADD	10.2
ASR	8.2
Logical operation	17.4
Branch	18
Compare	10.4
IT	20.2
LDR	8.4
LDM	13.8
STR	7.8
STM	10.2
SHIFT	5
MLA	4.4
MOV	14
REV	5.6
SUB	6.6
SXT	7.2
Register	7.2

6.2.4 Bus Test period

The bus startup test takes 88.8µs to execute.

Table 18 lists the bus periodic test periods. Measurement is carried out for each of the three separate test items (test numbers 1 to 3).

Table 18. Bus Periodic Test periods

TestNo	Periodic test (µs)
TestNo1	9.68
TestNo2	76
TestNo3	2.8

7 Reference Documents

1. IEC61508 1-7(ed2.0)
2. ARMv7-M Architecture Application Level Reference Manual, 2008
3. MB9BF504NB-DS706-00021-2v0-J.pdf (MB9B500 Data Sheet)
4. MB9B560R-DS709-00001.pdf(MB9B560R Data Sheet)
5. MB9Bxxx-MN706-0002-4v0-E (FM3 family Periphery Manual)
6. MN709-00001-1v0-J.pdf(FM4 family Periphery Manual)
7. Cortex-M3 Revision r2p1 Technical Reference Manual
8. Cortex-M4 Revision r0p1 Technical Reference Manual

8 Additional Information

For more information, please contact us on our website:

www.cypress.com/contact-us

Please mail us at:

customercare@cypress.com

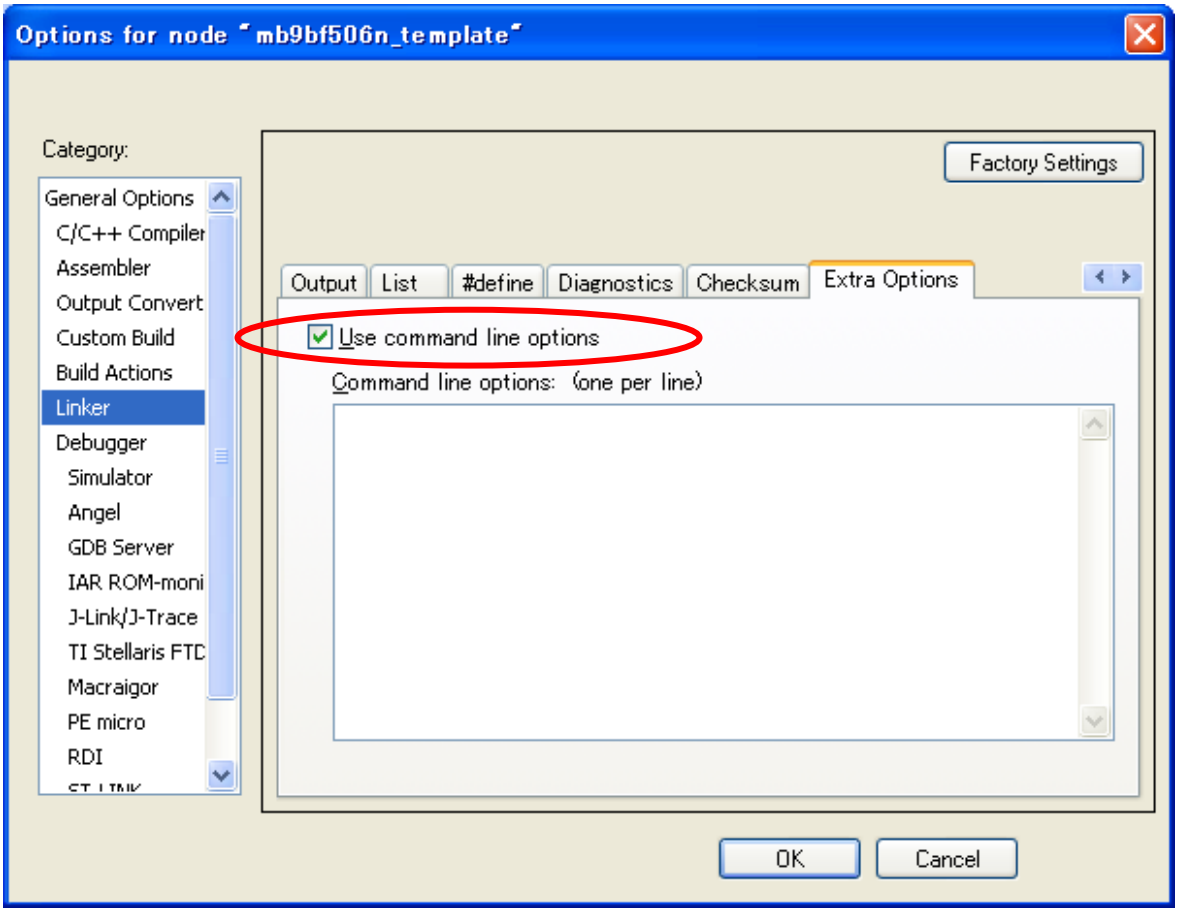
A Appendix

A.1 CRC code making method

The method to make CRC code to use in 5.1 ROM Test, follows is examples of IAR Embedded Workbench. Please refer to IAR's manual for details.

A.1.1 Start of the Command-Line

Click "Project"→"Options"→"Linker"→"Extra options" tabs, then check the "Use command line options".



A.1.2 Input the command

“--place_holder” command

"--place_holder" is used that make CRC code and a section in ROM. If input the following command, to set the size of section in 4byte and the alignment in 1.

--place_holder __checksum,4,.checksum,1

“--fill” command

The unused area of the target area needs to fill with optional value making the CRC code. Therefore, use “--fill” command. If input the following command, 0x00000000-0x00003FFF is filled with 0xFF.

--fill 0xFF;0x0000-0x3FFF

If input the following command, 0x00000000-0x00003FFF, 0x5000-0x5FFF and 0x6500-0x6FFF are filled with 0xFF.

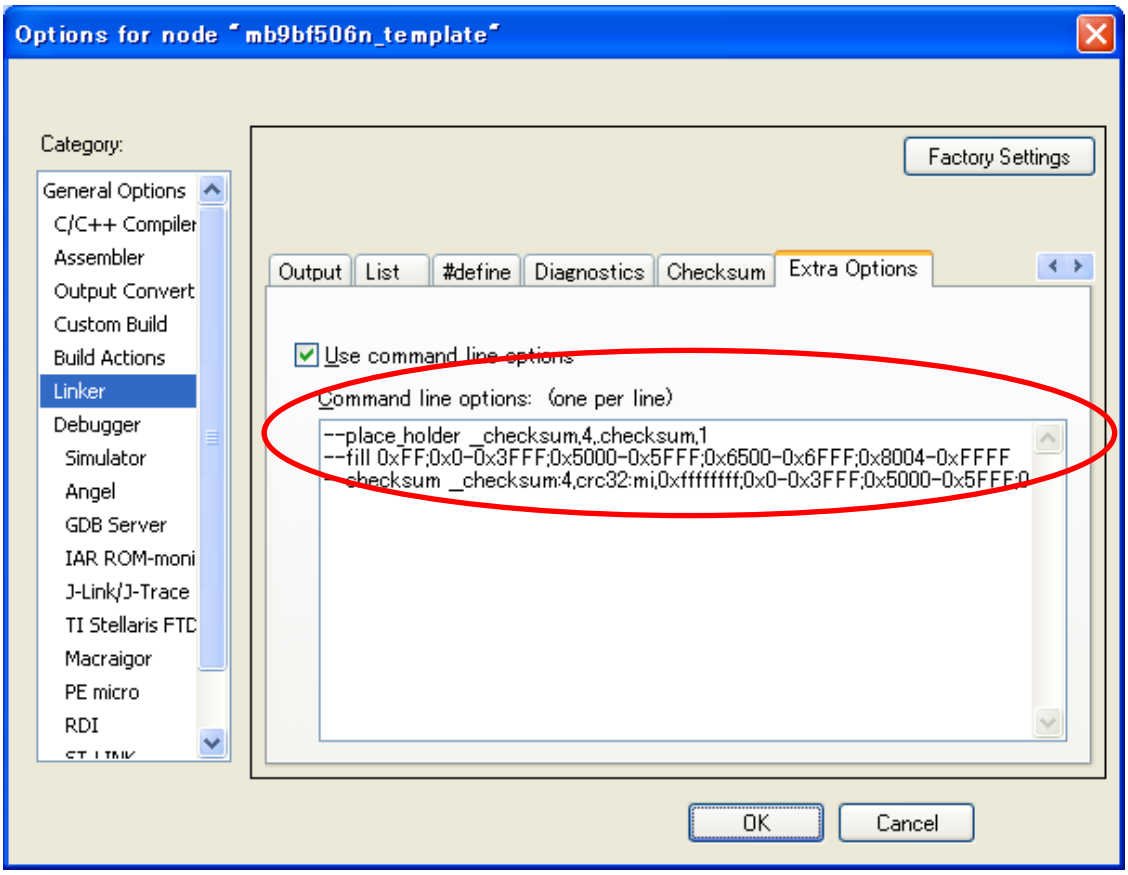
--fill 0xFF;0x0-0x3FFF;0x5000-0x5FFF;0x6500-0x6FFF

“--checksum” command

Set algorithm of CRC. If input the following command, you can set items as follow. The CRC code is stored in the symbol name “__checksum”, the CRC code size is 4byte, the algorithm is CRC32, calculation is LSB first, CRC code is initialized by 0xFFFFFFFF, 0x00000000-0x00003FFF, 0x5000-0x5FFF and 0x6500-x6FFF are filled with 0xFF.

--checksum __checksum:4,crc32:mi,0xffffffff;0x0-0x3FFF;0x5000-0x5FFF;0x6500-0x6FFF

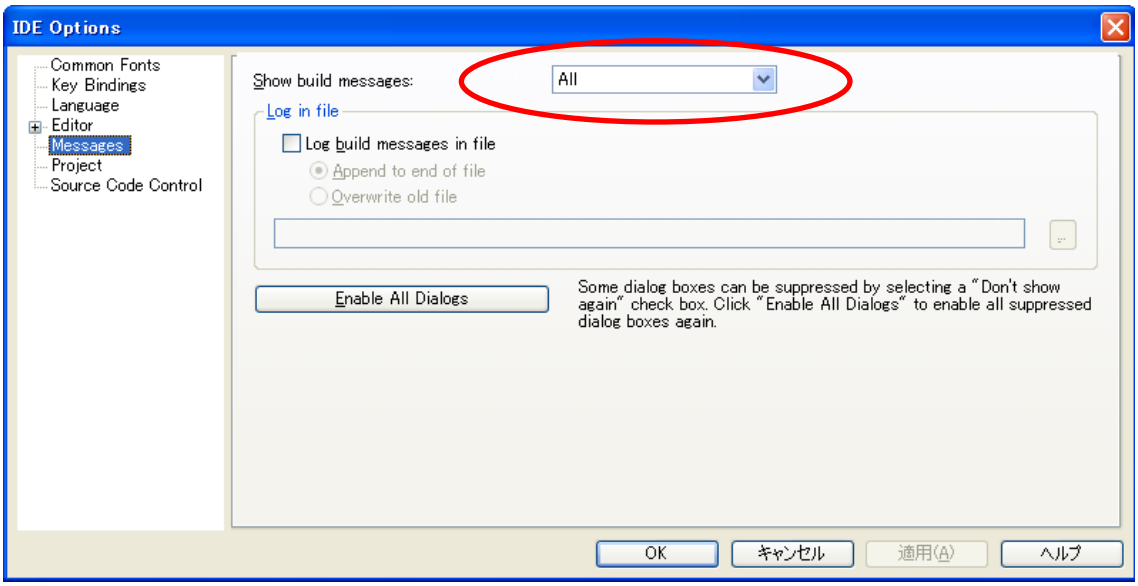
If input the command mentioned above (1, 2, and 3), close the window by clicking the “OK”.



A.1.3 Setting of build messages to display in the message window

If set the following contents, you can display build messages at the time of make to the message window.

Click “Tools”→”Options”→”Messages” tabs, then select the “All” from the combo box of “Show build messages”. Finally, close the window by clicking the “OK”.



A.1.4 Setting of the Linker configuration file

Add setting to the Linker configuration file to store CRC code in Flash. In the case of debug mode, you must use “mb9bf506_ram.icf” file. In the case of release mode, you must use “mb9bf506.icf” file. If input the following command, CRC code is stored in 0x8000.

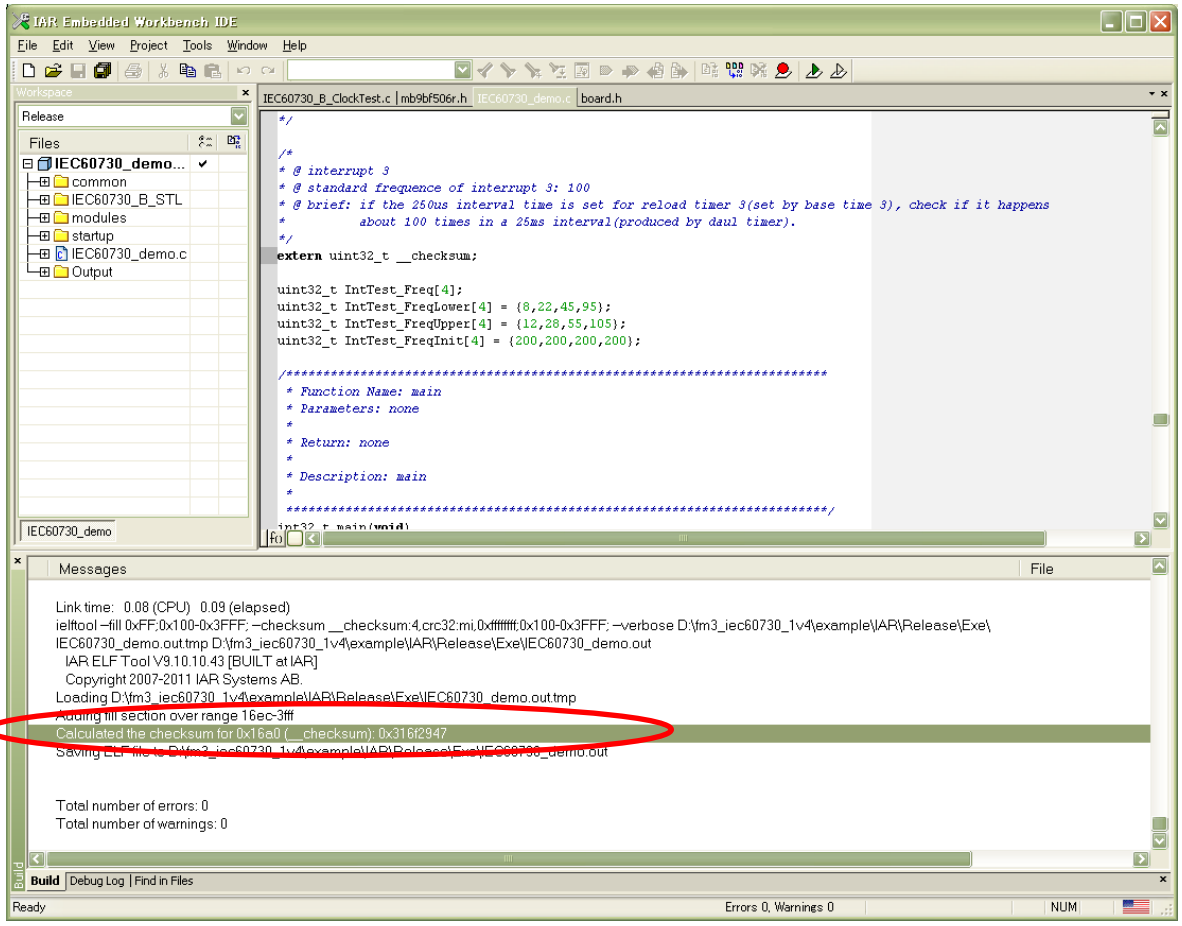
define symbol __ICFEDIT_checksum_start__ = 0x00008000;

define block CHECKSUM {ro section .checksum};

define symbol __ICFEDIT_checksum_start__ = 0x00008000;

A.1.5 Making CRC code

Confirm that the CRC code was made after make.



9 Document History

Document Title: AN204377 - FM3 and FM4 Family, IEC61508 SIL2 Self-Test Library

Document Number: 002-04377

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	YUIS	04/24/2014	Rev 1.0 Initial release
				Rev 2.0 Section-1:Porting to FM4 Section-5.3.1 and 5.3.2: Add FM4 DSP instruction and FPU instruction to CPU Test and Add DSTC bus test to BUS Test
*A	5099882	YUIS	07/14/2016	Migrated Spansion Application Note from FM3_AN706-00048-2v0-E to Cypress format.
*B	5824509	AESATMP9	07/19/2017	Updated logo and copyright.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709