User's Manual

R-IN32M3 Series

Programming Manual: Driver

- R-IN32M3-EC
- R-IN32M3-CL

Document number : R18UZ0009EJ0601
Issue date : Apr. 19, 2019

Renesas Electronics
www.renesas.com

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.

   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.
(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

General Precautions in the Handling of Products

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

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

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

3. Prohibition of Access to Reserved Addresses
   Access to reserved addresses is prohibited.
   - The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

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

How to Use This Manual

1. Purpose and Target Readers

This manual is intended for users who wish to understand the functions of industrial Ethernet network LSI "R-IN32M3-EC/CL" and design application systems using it.
Target users are expected to understand the fundamentals of electrical circuits, logic circuits, and microcomputers.

When designing an application system that includes this MCU, take all points to note into account. Points to note are given in their contexts and at the final part of each section, and in the section giving usage notes.

The list of revisions is a summary of major points of revision or addition for earlier versions. It does not cover all revised items. For details on the revised points, see the actual locations in the manual. The mark "<R>" in the text indicates the major revisions to this version. You can easily find these revisions by copying "<R>" and entering it in the search-string box for the PDF file.

Literature — Literature may be preliminary versions. Note, however, that the following descriptions do not indicate "Preliminary". Some documents on cores were created when they were planned or still under development. So, they may be directed to specific customers.

Documents related to the R-IN32M3 Series

Document Name	Document Number
R-IN32M3 Series Datasheet	R18DS0008EJ0200
R-IN32M3-EC User's Manual	R18UZ0003EJ0101
R-IN32M3-CL User's Manual	R18UZ0005EJ0101
R-IN32M3 Series User's Manual: Peripheral Modules	R18UZ0007EJ0301
R-IN32M3 Series Programming Manual: Driver	This manual
R-IN32M3 Series Programming Manual: OS	R18UZ0011EJ0300
R-IN32M3 Series User's Manual: TCP/IP Stack	R18UZ0019EJ0200

2. Numbers and Symbols

Data significance: Higher digits on the left and lower digits on the right

Active low representation:

xxxZ (capital letter Z after pin or signal name)

or xxx_N (capital letter _N after pin or signal name)

or xxnx (pin or signal name contains small letter n)

Note:

Footnote for item marked with Note in the text

Caution:

Information requiring particular attention

Remark:

Supplementary information

Numeric representation:

Binary … xxxx , xxxxB or n'bxxxx (n bits)

Decimal … xxxx

Hexadecimal … xxxxH or n'hxxxx (n bits)

Prefix indicating power of 2 (address space, memory capacity):

K (kilo) … $2^{10} = 1024$

M (mega) … $2^{20} = 1024^2$

G (giga) … $2^{30} = 1024^3$

Data Type:

Word … 32 bits

Halfword … 16 bits

Byte … 8 bits

Contents

List of Figures

List of Tables

1. Outline

Sample software code showing examples of the usage of each peripheral module of products of the R-IN32M3 series has been prepared to assist users in faster software development.
This document describes the operations of the drivers, specifications of the middleware and developer tool-dependent parts (startup routines, etc.) for the various peripheral modules of the R-IN32M3 series products.

1.1 Structure

The layered structure of sample software is shown below.

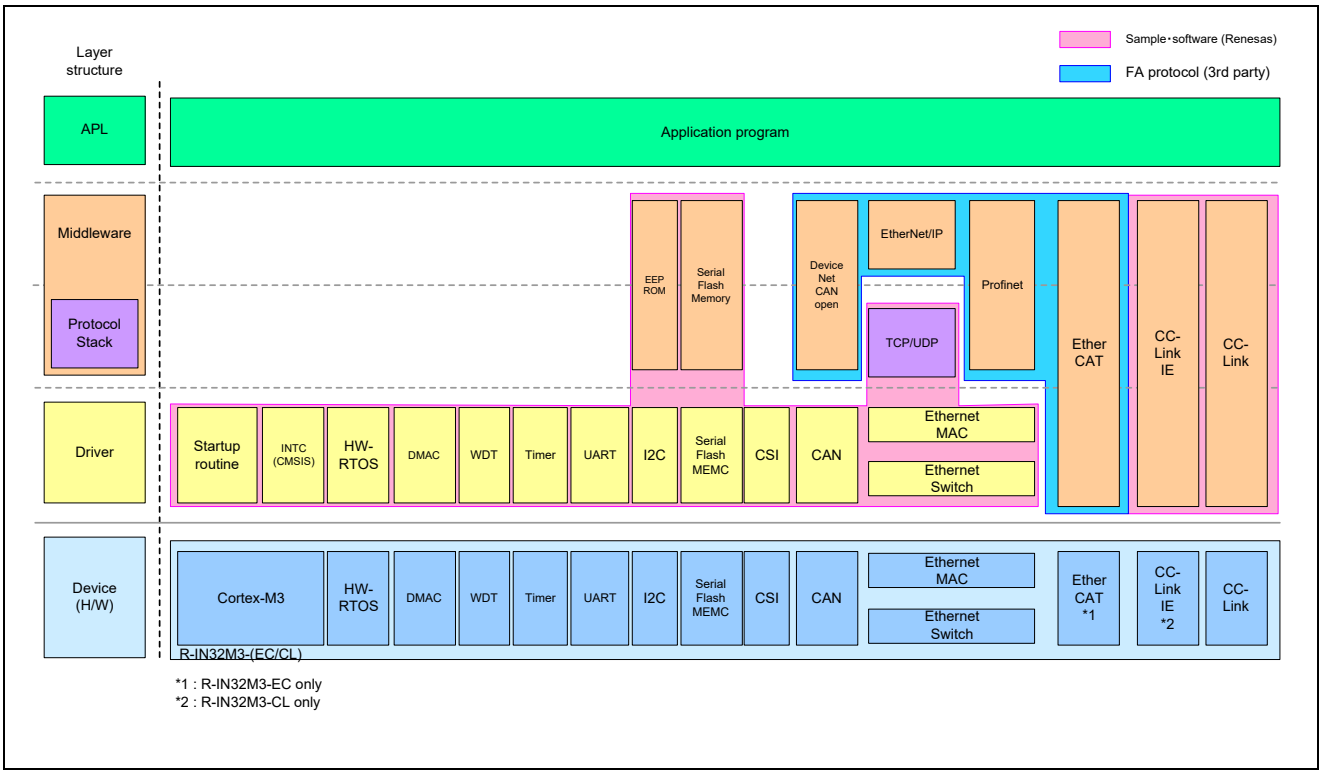

Figure 1.1 Layered Structure of the Sample Software

1.2 Development Environment

The sample software has been confirmed to work with the following tool chain.
This sample software adopts Arm® Cortex® Microcontroller Software Interface Standard (CMSIS) V2.10.
Regarding the detailed information, please refer to the documentation of CMSIS.

Table 1.1 List of Software Development Tools (Tool Chain)

Tool Chain	IDE	Compiler	Debugger	ICE
IAR	Embedded Workbench for Arm V6.60 ~ Latest Version (Please use the latest version) (IAR Systems)			I-jet JTAGjet-Trace-CM (IAR Systems)
Keil MDK-Arm	µVision V5.18.0.0 (Arm)			ULINK2 ULINKpro (Arm)
GNU	-	Sourcery G++ Lite for ARM EABI 2012.09-63 (Mentor Graphics)	microVIEW-PLUS Ver.5.11PL3 (DTS INSIGHT <R>)	adviceLUNA 2.03-00 (DTS INSIGHT <R>)

Table 1.2 List of Software Development Tools (Development Environment)

Type of Tool	Product Name	Version	Tool Vendors
Development environment	cygwin[Note]	-	Red Hat
Make tool	GNU make (for cygwin[Note])	3.81	GPL

Note: Please access http://cygwin.com/ for the installation of cygwin.
Please also install "GNU make" when you install cygwin.

2. Configuration of Files

This section lists the configuration of directories and files for the sample software.

2.1 Configuration of Directories

Table 2.1 Configuration of Directories for Sample Software

Directory	Contents
./	Directory of sample software
./CMSIS	Directory of Arm Cortex-M3 related-definition files (CMSIS is used as it is)
./Device	Directory of device specific files
./ Device /Renesas/RIN32M3	Directory of R-IN32M3 specific programs
./ Device /Renesas/RIN32M3/Include	Directory of include files
./ Device /Renesas/RIN32M3/Library	Directory of library files
./ Device /Renesas/RIN32M3/Source	Directory of source files

2.2 ./ Device /Renesas/RIN32M3/Include: Include Files

The configuration of include files is listed below.

Table 2.2 Configuration of Files in Include File Directories

Directory	File	Contents
csi/	csi.h	Prototype declaration of CSI driver
iic/	iic.h	Prototype declaration of I2C driver
sromc/	sromc.h	Prototype declaration of serial flash ROM driver
timer/	timer.h	Prototype declaration of timer driver
uart/	uart.h	Prototype declaration of UART driver
hwos/	hwos_hwfnc.h	HW-RTOS driver header file
wdt/	wdt.h	Prototype declaration of watchdog timer driver
dmac/	dmac.h	DMA driver header file
./	errcodes.h	Error definition file
./	itron.h	ITRON general definition file (data type, constant value, macro)
./	kernel.h	Main function definition file (service call, data type, constant value, macro)
./	RIN32M3.h	Include header for R-IN32M3 series
./	RIN32M3_EC.h	Device definition file for R-IN32M3-EC
./	RIN32M3_CL.h	Device definition file for R-IN32M3-CL
./	system_RIN32M3.h	System information definition based on CMSIS

2.3 ./ Device /Renesas/RIN32M3/Library: Library Files

The configuration of library files is listed below.

Table 2.3 Configuration of Files in Library Directories

Directory	File	Contents
ARM/	libos.a	H/W-RTOS driver library
GCC/	libos.a	H/W-RTOS driver library
IAR/	libos.a	H/W-RTOS driver library

2.4 ./ Device /Renesas/RIN32M3/Source: Source Files

The configuration of source directories is listed below.

Table 2.4 Configuration of Source Directories

Directory	Contents
Driver	Drivers
Middleware	Middleware
Project	Sample application
Template	Startup files

2.4.1 ./ Device /Renesas/RIN32M3/Source/Driver: Driver Files

The configuration of driver source files is listed below.

Table 2.5 Configuration of Files in Driver Directories

Directory	File	Contents
csi/	csi.c	CSI driver
dmac/	dmac.c	DMAC driver
iic/	iic.c	IIC driver
sromc/	sromc.c	Serial flash ROM driver
timer/	timer.c	Timer driver
uart/	uart.c	UART driver
wdt/	wdt.c	Watchdog time driver

2.4.2 ./ Device /Renesas/RIN32M3/Source/Middleware: Middleware

The configuration of source files for the middleware is listed below.

Table 2.6 Configuration of Files in Middleware Directories

Directory	File	Contents
eeprom/	eeprom.c	EEPROM middleware sample source
	eeprom.h	EEPROM middleware header file
flash/	flash.c	Parallel flash ROM middleware sample source
	flash.h	Parallel flash ROM middleware header file
sflash/	sflash.c	Serial flash ROM middleware sample source
	sflash.h	Serial flash ROM middleware header file

2.4.3 ./ Device /Renesas/RIN32M3/ Source/Project: Sample Application

The configuration of the sample applications is listed below.
Regarding the outline of operation of the sample application, refer to section 8, Example of Application.

Table 2.7 Configuration of Files in Directories for the Sample Applications

Directory	File	Contents
osless_sample/	main.c	Main process source file
osless_sample/ARM/	main.uvoptx	Debugger (MDK-Arm) related file
	main.uvprojx	Debugger (MDK-Arm) project file
	Makefile	Makefile
	scat_boot_extrom.ld	Linker scripts (boot code: Flash ROM allocated)
	scat_boot_iram.ld	Linker scripts (boot code: Instruction RAM allocated)
	scat_boot_sflash.ld	Linker scripts (boot code: Serial flash ROM allocated)
osless_sample/GCC/	Makefile	Makefile
	scat_boot_extrom.ld	Linker scripts (boot code: Flash ROM allocated)
	scat_boot_iram.ld	Linker scripts (boot code: Instruction RAM allocated)
	scat_boot_sflash.ld	Linker scripts (boot code: Serial flash ROM allocated)
	rin32m3ec.mvp	Debugger (adviceLUNA) project file
	rin32m3ec.mvr	Debugger (adviceLUNA) related file
osless_sample/IAR/	main.eww	IAR project file
	main.ewd	IAR project related file
	main.ewp	IAR project related file
	boot_norflash.icf	Linker scripts (boot code: Flash ROM allocated)
	iram.icf	Linker scripts (boot code: Instruction RAM allocated)
	boot_serialflash.icf	Linker scripts (boot code: Serial flash ROM allocated)
	init.mac	Debugger macro script file

2.4.4 ./ Device /Renesas/RIN32M3/ Source/Templates: Startup Files

The configuration of source files such as startup files is listed below.

Table 2.8 Configuration of Files in Startup Directories

Directory	File	Contents
Templates/ARM/	startup_RIN32M3.c	Startup file (for MDK-Arm)
	syscalls.c	To replace library functions (for MDK-Arm)
Templates/GCC/	startup_RIN32M3.c	Startup file (for GCC)
	syscalls.c	To replace library functions (for GCC)
Templates/IAR/	cstartup_M.c	Startup file (for IAR)
	vectors_M.c	Vector definition file
	vectors_rom.c	Vector definition file
	syscalls.c	To replace library functions (for IAR)
Templates/	system_RIN32M3.c	Startup file (common)

3. Software Development Procedure

In this section, a series of procedures for software development is explained.

3.1 Design Flow

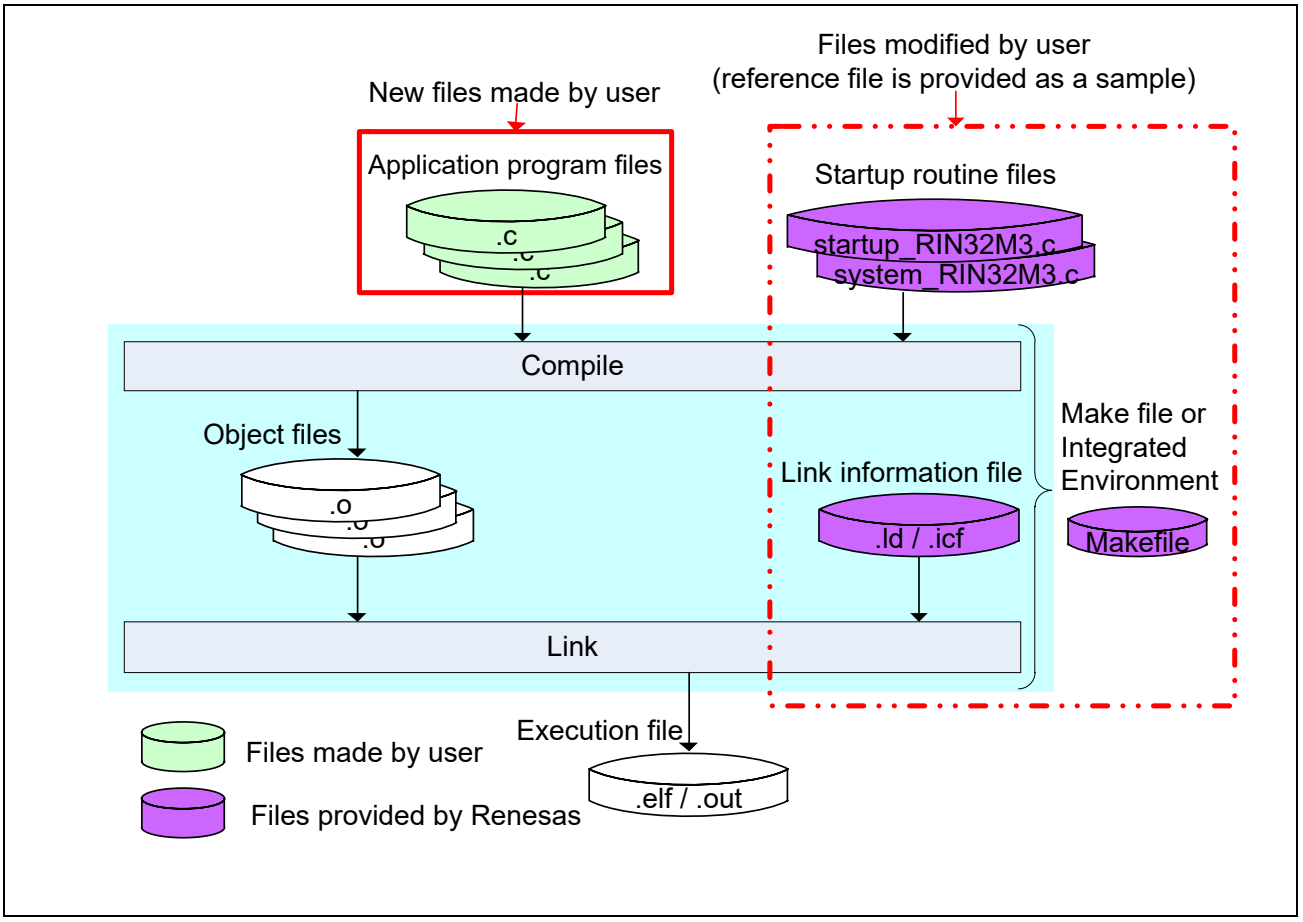

Figure 3.1 File Correlation Diagram

3.2 Memory Maps

3.2.1 Memory Maps

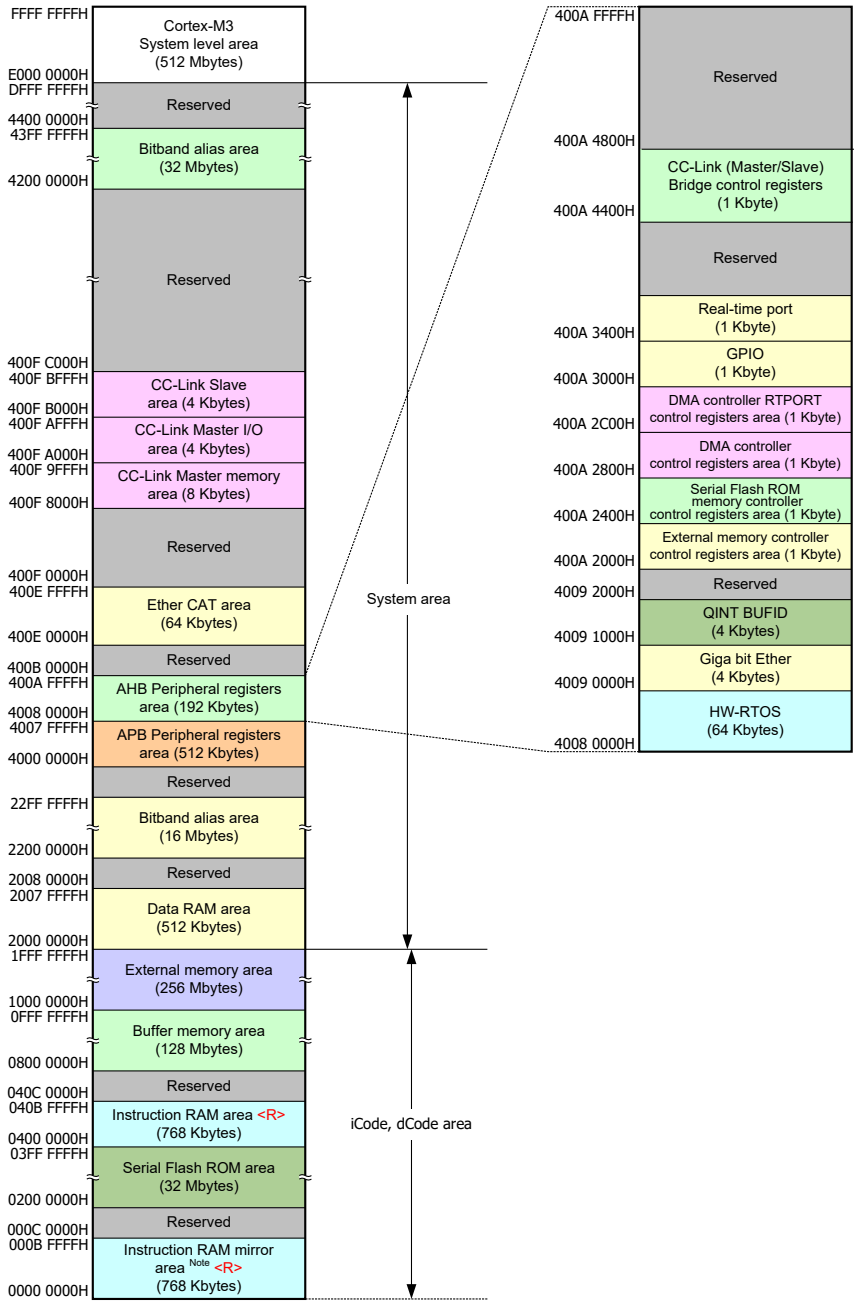

Figure 3.2 Entire Memory Map (R-IN32M3-EC)

<R>Note: The addresses of the instruction RAM mirror area (768 Kbytes) where access actually occurs will change according to the selected boot mode. For details, see section 5.3, Memory MAP in Each Boot Mode, in the R-IN32M3 Series User's Manual: Peripheral Modules.

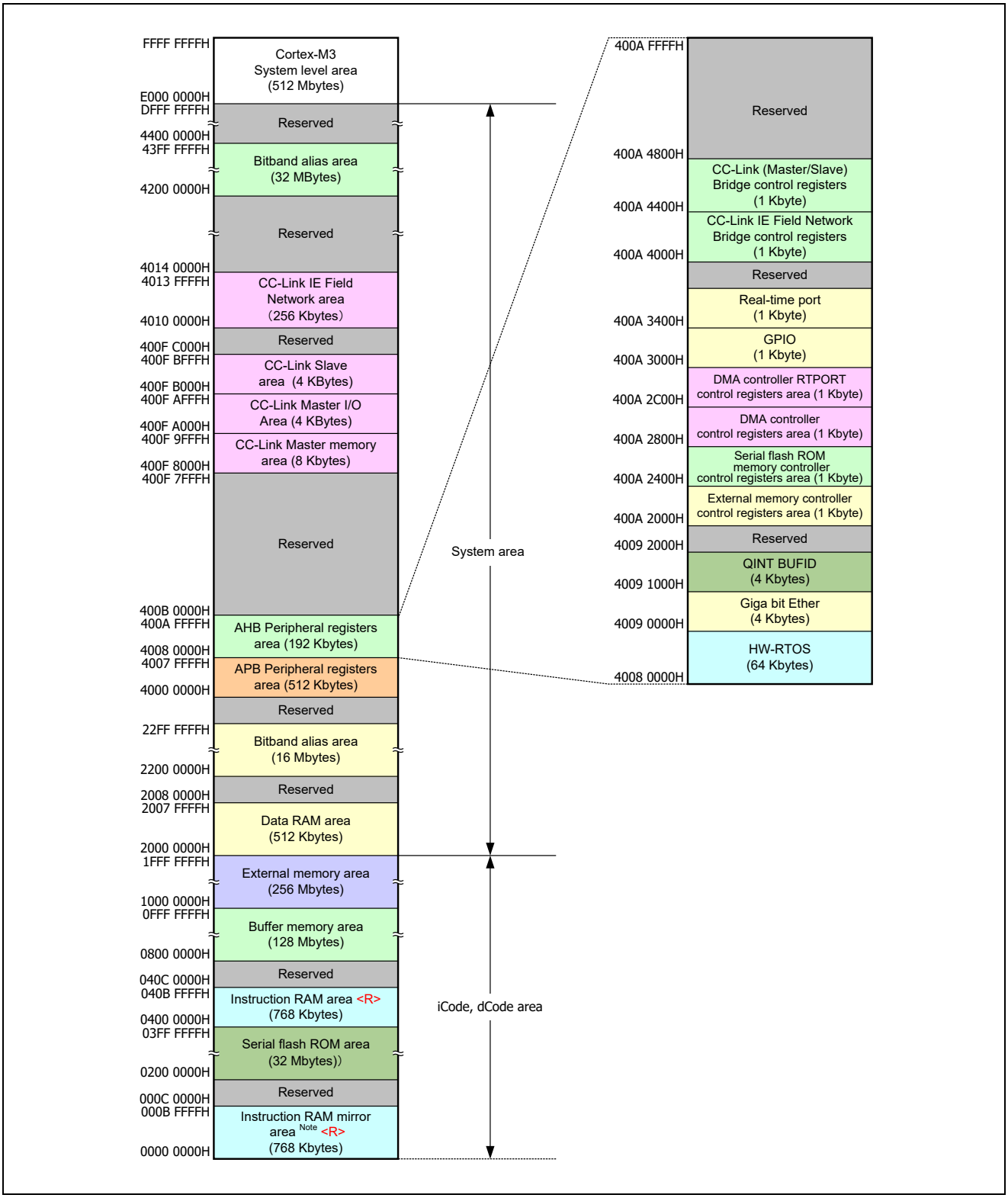

Figure 3.3 Entire Memory Map (R-IN32M3-CL)

<R>Note: The addresses of the instruction RAM mirror area (768 Kbytes) where access actually occurs will change according to the selected boot mode. For details, see section 5.3, Memory MAP in Each Boot Mode, in the R-IN32M3 Series User's Manual: Peripheral Modules.

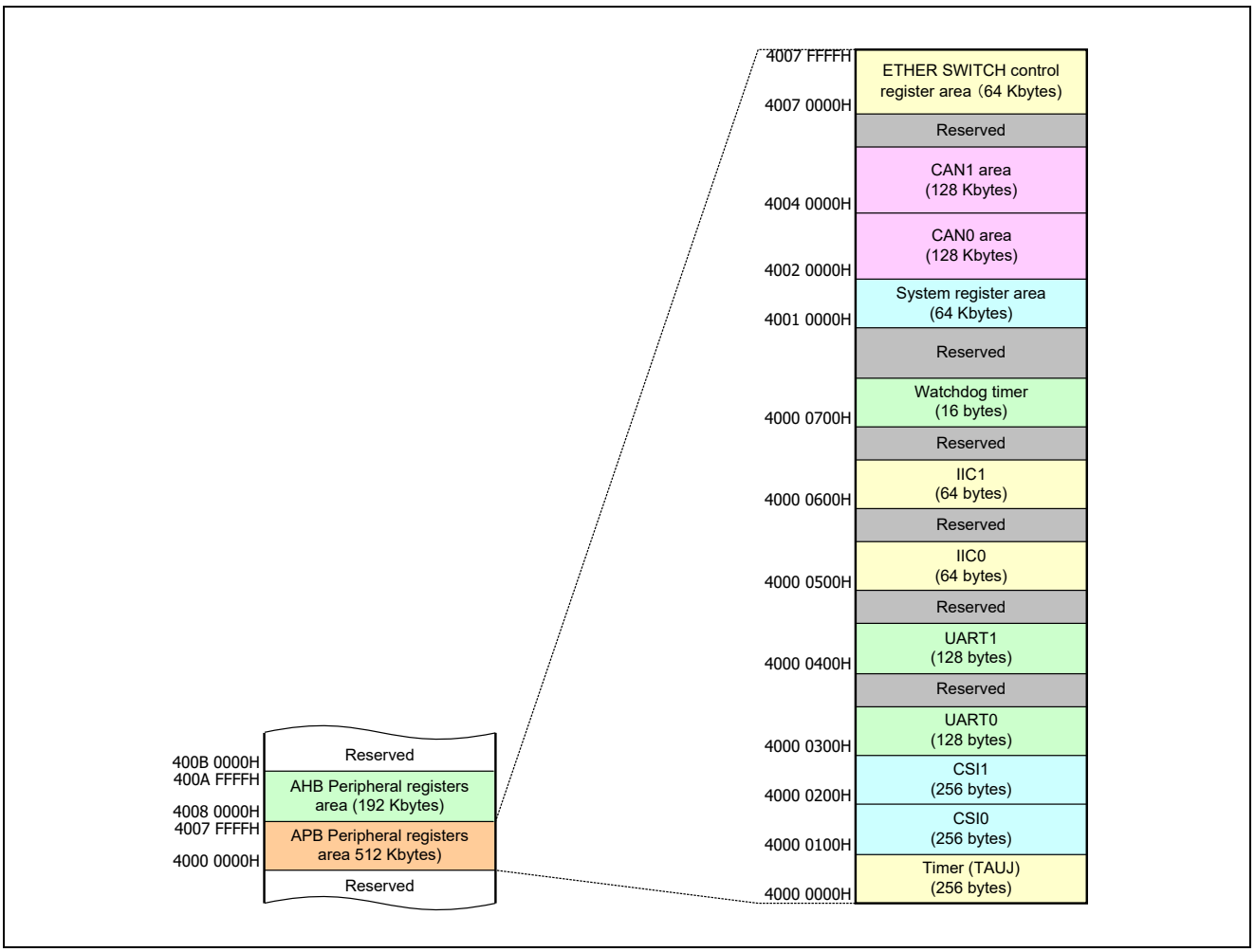

Figure 3.4 Memory Map (APB Peripheral Registers Area) (common to R-IN32M3-EC/CL)

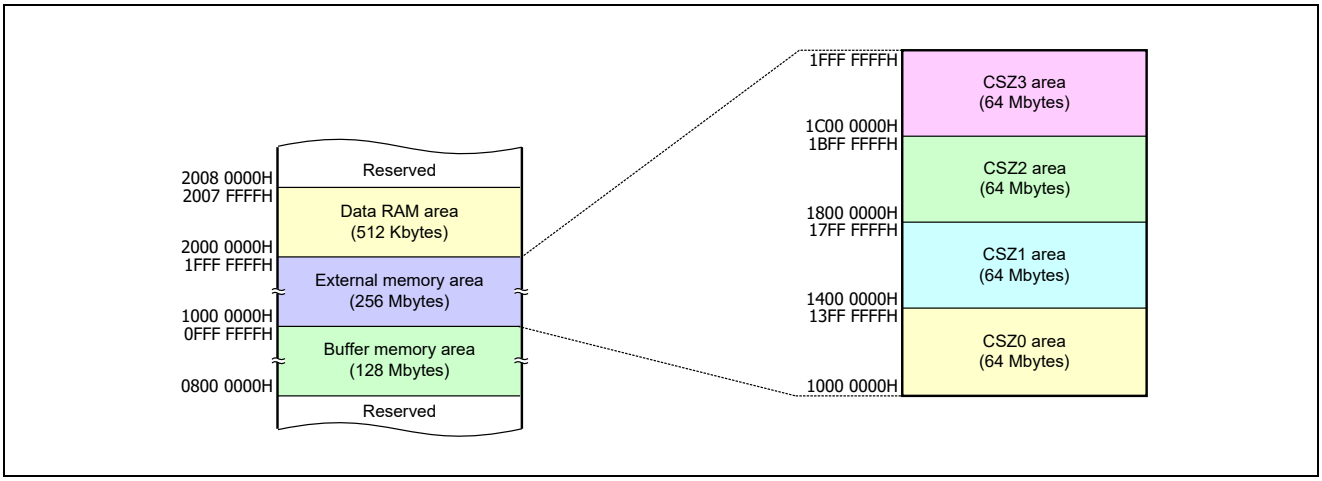

Figure 3.5 Memory Map (External Memory Area) (common to R-IN32M3-EC/CL)

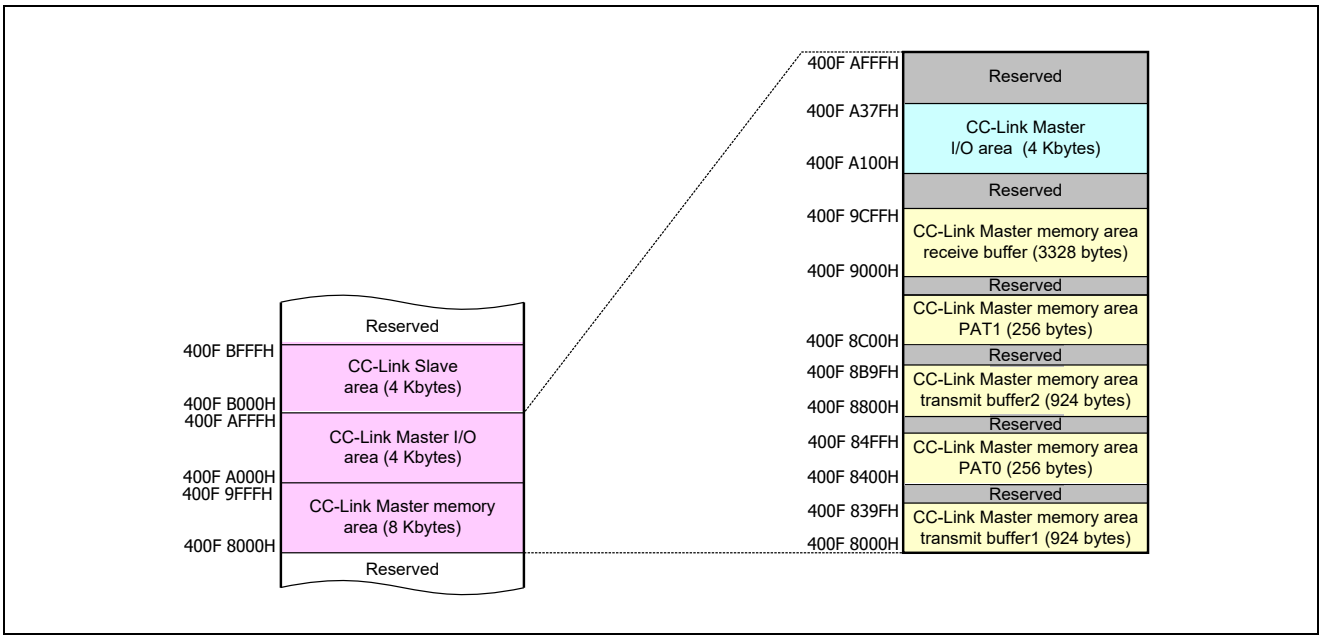

Figure 3.6 Memory Map (CC-Link Master Area) (common to R-IN32M3-EC/CL)

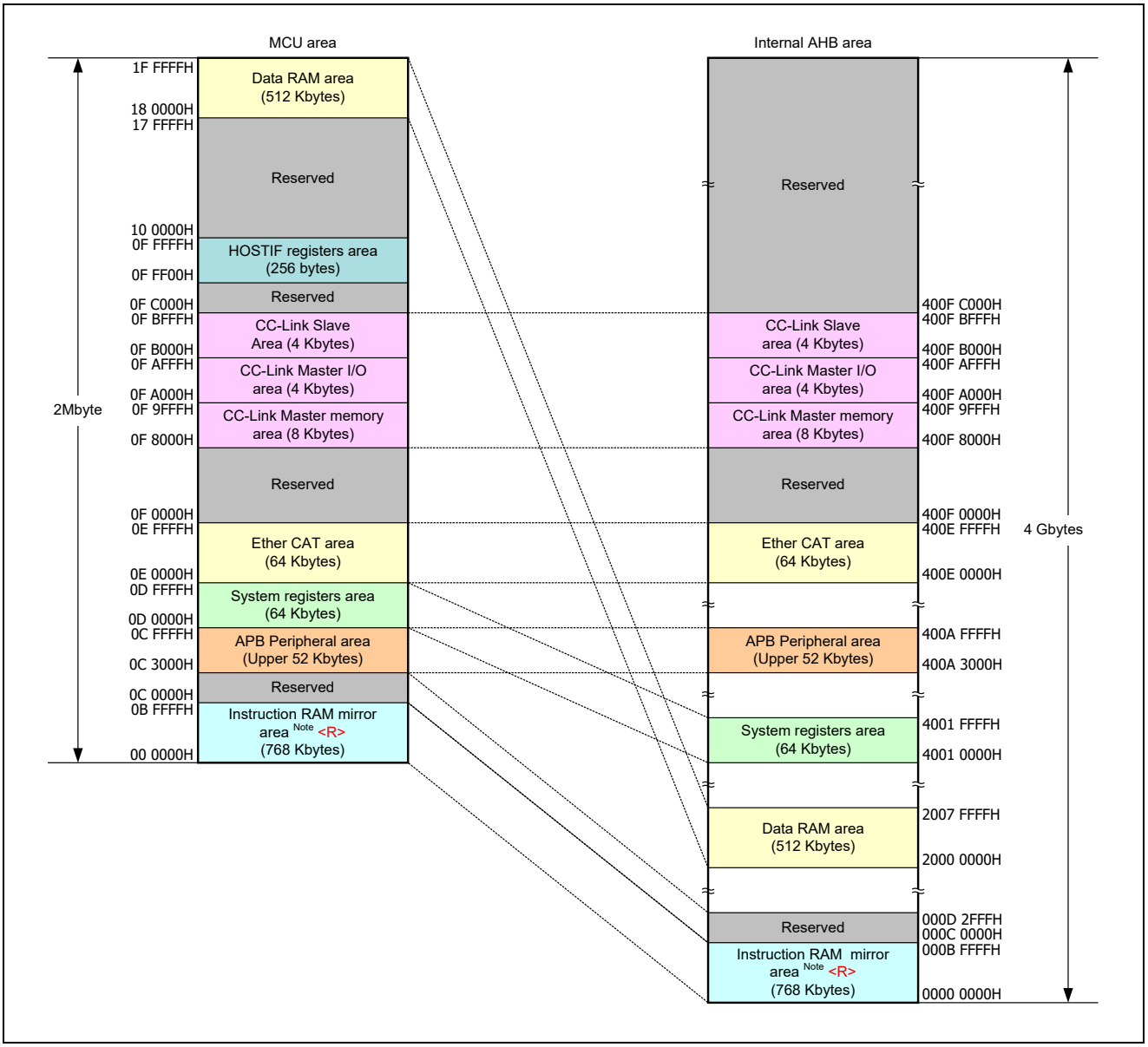

Figure 3.7 External MCU Interface Space (R-IN32M3-EC)

<R>Note: The addresses of the instruction RAM mirror area (768 Kbytes) where access actually occurs will change according to the selected boot mode, as shown in the table below. For details, see section 5.3, Memory MAP in Each Boot Mode, and section 4, Bus Architecture, in the R-IN32M3 Series User's Manual: Peripheral Modules.

BOOT1	BOOT0	Boot Mode	Access Destination Area	Remarks
0	0	External memory boot	—	External MCU interface is disabled
0	1	External serial flash ROM boot	Reserved	Access disabled
1	0	External MCU boot	Instruction RAM area	—
1	1	Instruction RAM boot	Instruction RAM area	Enabled only for debugging

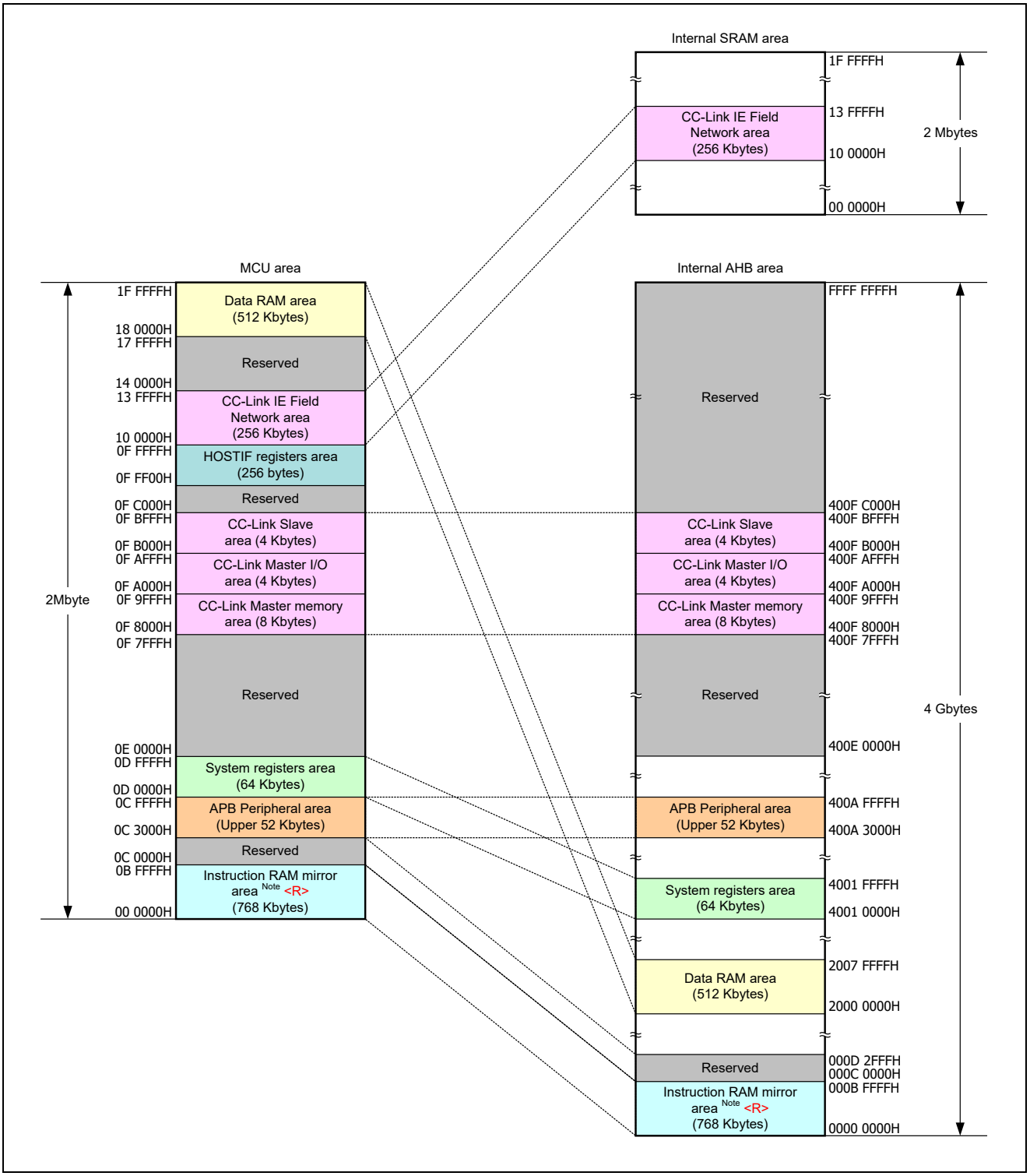

Figure 3.8 External MCU Interface Space (R-IN32M3-CL)

<R>Note: The addresses of the instruction RAM mirror area (768 Kbytes) where access actually occurs will change according to the selected boot mode, as shown in the table below. For details, see section 5.3, Memory MAP in Each Boot Mode, and section 4, Bus Architecture, in the R-IN32M3 Series User's Manual: Peripheral Modules.

BOOT1	BOOT0	Boot Mode	Access Destination Area	Remarks
0	0	External memory boot	—	External MCU interface is disabled
0	1	External serial flash ROM boot	Reserved	Access disabled
1	0	External MCU boot	Instruction RAM area	—
1	1	Instruction RAM boot	Instruction RAM area	Enabled only for debugging

3.2.2 Program Allocation

The figure below shows an example of program allocation when booting is from the parallel flash ROM.

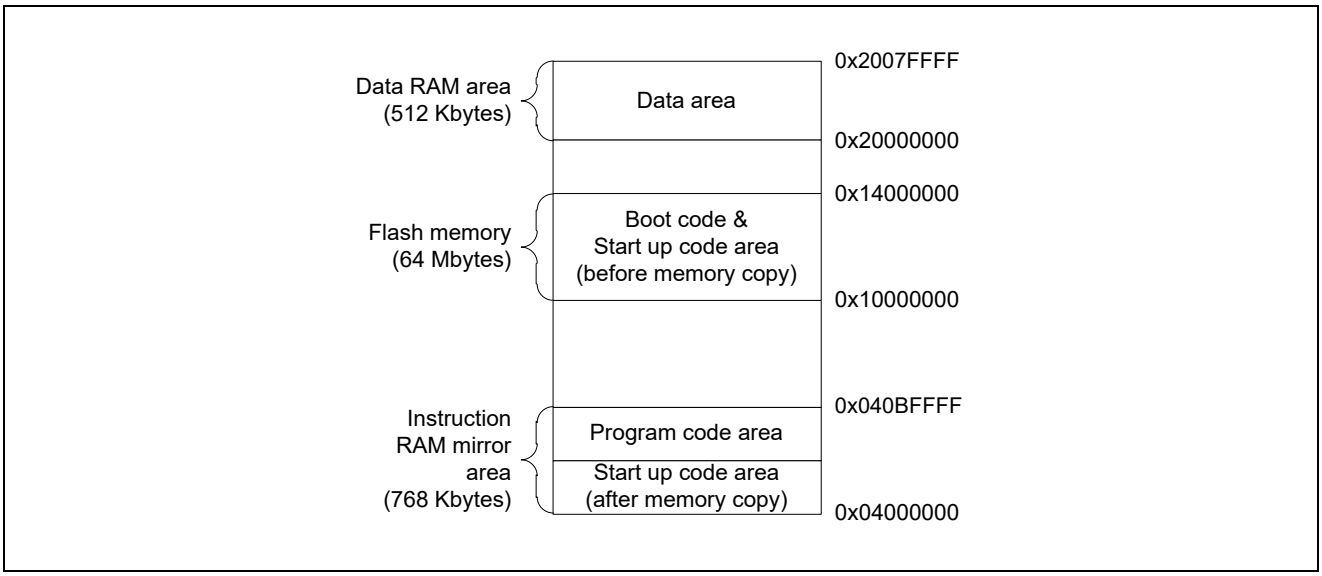

Figure 3.9 Program Allocation

4. Data Type and Macro

This section explains data type and macro in this sample software.

4.1 Data Type

The table below lists data types defined by the sample software.

Table 4.1 Data Type

Macro	Type	Meaning
ER_RET	int	Error code used as the returned value of a function
IRQn	enum	Interrupt number

4.2 Macro Definition

This section lists the definitions of this sample software.

4.2.1 Constants

The constants defined in this sample software are as follows.

Table 4.2 Constant (General)

Constant	Value	Meaning
NULL	((void*))0	Invalid pointer

Table 4.3 Constants (System)

Constant	Value	Meaning
RIN32M3_SYSCLK	100000000	Clock frequency of the system (unit: Hz)
SYS_UART_CH	1	UART channel number in the system

Table 4.4 Constant (Error Code)

Constant	Value	Meaning
ER_OK	0	Normal end
ER_NG	-1	Abnormal end
ER_SYS	-2	Undefined error
ER_PARAM	-3	Detection of an invalid parameter
ER_NOTYET	-4	Process incomplete
ER_NOMEM	-5	Out of the memory range
ER_BUSY	-6	Busy state
ER_INVAL	-7	Invalid state
ER_TIMEOUT	-8	Timeout occurred

4.2.2 Definitions for Conditional Compilation

The macro definitions for use in conditional compilation are listed below.

Table 4.5 Macro Definitions for Conditional Compilation

Definition Name	Contents	Definition File
OSLESS	Determine whether HW-RTOS is used or not If defined: HW-RTOS is not used If not defined: HW-RTOS is used	startup_RIN32M3.c cstartup_M.c
RIN32M3_CL	To show that R-IN32M3-CL is used	RIN32M3.h system_RIN32M3.c flash.h

Caution: If you are using the R-IN32M3-CL, please define "RIN32M3_CL" before compilation. If the "RIN32M3_CL" macro is not defined, sample software generates codes for the R-IN32M3-EC.

5. Definitions of R-IN32M3 Registers (RIN32N3.h)

The include file RIN32M3.h defines interrupts and registers of the R-IN32M3-EC or R-IN32M3-CL. Table 5.1 and Table 5.2 list definitions of the registers. For details of the individual registers, see the relevant section of the document under "Reference of Register Details" in each table.

Caution: If you are using the R-IN32M3-CL, please define "RIN32M3_CL" before compilation. If the "RIN32M3_CL" macro is not defined, sample software defines interrupts and registers for the R-IN32M3-EC.

5.1 APB Peripheral Registers

Table 5.1 Definitions of APB Peripheral Registers

#define	Function	Reference of Register Details
#define RIN_TMR_BASE	32-bit timer register (TAUJ2)	Section 14 of the *R-IN32M3 Series User's Manual: Peripheral Modules*.
#define RIN_CSI0_BASE	CSI channel 0 register	Section 17 of the *R-IN32M3 Series User's Manual: Peripheral Modules*.
#define RIN_CSI1_BASE	CSI channel 1 register	
#define RIN_UART0_BASE	UART channel 0 register	Section 16 of the *R-IN32M3 Series User's Manual: Peripheral Modules*.
#define RIN_UART1_BASE	UART channel 1 register	
#define RIN_IIC0_BASE	I2C channel 0 register	Section 18 of the *R-IN32M3 Series User's Manual: Peripheral Modules*.
#define RIN_IIC1_BASE	I2C channel 1 register	
#define RIN_WDT_BASE	Watchdog timer register	Section 15 of the *R-IN32M3 Series User's Manual: Peripheral Modules*.
#define RIN_SYS_BASE	System register	Relevant section of the *R-IN32M3 Series User's Manual: Peripheral Modules*.
#define RIN_CAN0_BASE	CAN channel 0 register	Section 19 of the *R-IN32M3 Series User's Manual: Peripheral Modules*.
#define RIN_CAN1_BASE	CAN channel 1 register	
#define RIN_ETHSW_BASE	Ethernet switch register	Section 8 of the *R-IN32M3 Series User's Manual: Peripheral Modules*.

5.2 AHB Peripheral Registers

Table 5.2 Definitions of AHB Peripheral Registers

Code	Function	Reference of Register Details
#define RIN_HWOS_BASE	HW-RTOS register	Section 7 of the *R-IN32M3 Series User's Manual: Peripheral Modules*.
#define RIN_ETH_BASE	Gigabit Ethernet MAC register	Section 7 of the *R-IN32M3 Series User's Manual: Peripheral Modules*.
#define RIN_MEMC_BASE	Asynchronous SRAM memory controller register	Section 9 of the *R-IN32M3 Series User's Manual: Peripheral Modules*.
#define RIN_SROM_BASE	Serial flash ROM memory controller register	Section 12 of the *R-IN32M3 Series User's Manual: Peripheral Modules*.
#define RIN_DMAC0_BASE	DMAC channel 0 register	Section 13 of the *R-IN32M3 Series User's Manual: Peripheral Modules*.
#define RIN_DMAC1_BASE	DMAC channel 1 register	
#define RIN_DMAC2_BASE	DMAC channel 2 register	
#define RIN_DMAC3_BASE	DMAC channel 3 register	
#define RIN_RTDMAC_BASE	RTDMAC register	
#define RIN_GPIO_BASE	Port register	Section 7 of the *R-IN32M3 Series User's Manual R-IN32M3-CL* or section 8 of the *R-IN32M3-CL Series User's Manual: R-IN32M3-EC*.
#define RIN_RTPORT_BASE	RT port register	
#define RIN_CCI_BRG_BASE[Note 1]	CC-Link IE Field bridge control register	Section 6 of the *R-IN32M3 Series User's Manual: R-IN32M3-CL*.
#define RIN_CC_BR_BASE	CC-Link bridge control register	Section 20 of the *R-IN32M3 Series User's Manual: Peripheral Modules*.
#define RIN_SMC_BASE	Synchronous burst access memory controller register	Section 10 of the *R-IN32M3 Series User's Manual: Peripheral Modules*.
#define RIN_ECAT_BASE[Note 2]	EtherCAT register	Section 6 of the *R-IN32M3-CL Series User's Manual: R-IN32M3-EC*.
#define RIN_CCLSLAVE_BASE	CC-Link remote device station register	*R-IN32 Series User's Manual: CC-Link Remote Device Station*

Notes 1. This is only available when the R-IN32M3-CL is used.
2. This is only available when the R-IN32M3-EC is used.

6. Driver

This section explains the functions of the drivers.

6.1 List of Driver Functions

The API functions in this sample software are listed below.

Table 6.1 Timer Driver Functions

Function Name	Summary
clock_init	Initialization of system timer (channel 0)
timer_interval_init	Initialization of interval timer
timer_onecount_hwtrg_init	Initialization of one-count timer (triggered by hardware)
timer_start	Start timer
timer_stop	Stop timer
timer_check_act	Check the timer activation

Table 6.2 UART Driver Functions

Function Name	Summary
uart_init	Initialization of UART
uart_write	Transmit one byte of character data
uart_read	Receive one byte of character data
uart_check_receivedata	Check presence of received data

Table 6.3 IIC Driver Functions

Function Name	Summary
iic_init	Initialization of IIC controller
iic_start_condition	Transmit start condition
iic_stop_condition	Transmit stop condition
iic_write	Transmit one byte of character data
iic_read	Receive one byte of character data

Table 6.4 CSI Driver Functions

Function Name	Summary
csi_init	Initialization of CSI controller
csi_write	Transmit one byte of character data
csi_read	Receive one byte of character data
csi_check_tx	Check transmission data (for slave)
csi_check_rx	Check received data (for slave)
csi_change_mode	Tx/Rx mode change (for slave)

Table 6.5 DMAC Driver Function

Function Name	Summary
dmac_memcpy	Copy memory (DMA transfer)

Table 6.6 Serial Flash ROM Driver Functions

Function Name	Summary
sromc_init	Initialization of SPI bus controller
sromc_write	Write data to SPI bus
sromc_read	Read data form SPI bus

Table 6.7 Watchdog Timer Driver Functions

Function Name	Summary
wdt_init	Initialization of watchdog timer
wdt_start	Watchdog timer activation
wdt_clear	Counter clearing
wdt_wait_reset	Wait for reset

Table 6.8 CAN Driver Functions<R>

Function Name	Summary
can_enable	
can_init	
can_shutdown	
can_get_mode	
can_set_mode	
can_get_id_data_dlc	
can_get_data_dlc	
can_set_id_data_dlc	
can_set_data	
can_tx_req	
can_get_txinfo	
can_get_rxinfo	
can_get_ch_status	
can_clr_ch_status	
can_get_bus_staus	

6.2 Timer Control

6.2.1 Initialization of Timer Module

clock_init

(1) Description

Initialization of the timer module

(2) C-Language Format

```
void clock_init( void );
```

(3) Parameter

None

(4) Function

This function initializes the system timer to use the "clock" function.

(5) Return Value

None

Remark: This API uses "TAUJ2TTINm input position detection" described in Table 14.3, in the R-IN32M3 Series User's Manual: Peripheral Modules.

6.2.2 Initialization of Interval Timer

timer_interval_init

(1) Description

Initialization of the interval timer

(2) C-Language Format

```
ER_RET timer_interval_init( uint8_t ch, uint_32t i_time );
```

(3) Parameter

I/O	Parameter	Description
I	uint8_t ch	Channel selection argument 0: channel- 0 1: channel- 1 2: channel- 2 3: channel- 3
I	uint32_t i_time	Periodic time interval (1 to 42,949 ms)

(4) Function

This function sets the timer selected by the channel selection argument to interval timer mode.
Although the interval timer outputs an interrupt signal in a cycle given by the interval argument, this interrupt is assigned the lowest priority in this sample program.
A parameter error is returned if the channel selection argument is other than 0 to 3 or the interval argument is other than 1 to 42949.

- Timer clock specification is as follows.
  - > Reference clock (PCLK) frequency: 100 MHz
  - > Count clock frequency: 100 MHz

(5) Return Value

Return Value	Meaning
ER_OK	Initialization succeeded
ER_PARAM	Parameter error - The selected channel is NOT 0 to 3 - The interval time is NOT 1 to 42,949 ms

Remark: This API uses "Interval timer" described in Table 14.3, TAUJ2 Operations in the R-IN32M3 Series User's Manual: Peripheral Modules.

6.2.3 Initialization of One-Count Timer (triggered by hardware)

timer_onecount_hwtrg_init

(1) Description

Initialization of the one-count timer (triggered by hardware)

(2) C-Language Format

```
ER_RET timer_onecount_hwtrg_init( uint8_t ch,uint32_t o_time, uint32_t trg );
```

(3) Parameter

I/O	Parameter		Description
I	uint8_t	ch	Channel selection argument 0: channel- 0 1: channel- 1 2: channel- 2 3: channel- 3
I	uint32_t	o_time	Time for counting once (10 to 4,294,967,285 ns)
I	uint32_t	trg	Trigger source selection argument (the number of IRQ in trigger sources +4)

(4) Function

This function sets the timer selected by the channel selection argument to one-count timer mode. This timer is triggered by the interrupt signal which is selected by the trigger source selection argument.
The timer stops counting after the specific time given by the "time for counting once" argument has elapsed.
The timer does not detect a trigger during counting.
A parameter error is returned if the value of the channel selection argument or the time for counting once is not available.
Set the frequency of the clock to drive counting by the timer to 100 MHz for this operation.

Caution: The interrupt will not be detectable if the counter clock period is longer than the interrupt pulse width.

(5) Return Value

Return Value	Meaning
ER_OK	Initialization succeeded
ER_PARAM	Parameter error - The selected channel is NOT 0 to 3 - The time for counting once is NOT 1 to 42,949,672 ns

Remark: This API uses "delay counting" described in "TAUJ2 Operations" in the R-IN32M3 Series User's Manual: Peripheral Modules.

6.2.4 Starting Timer

timer_start

(1) Description

Starting the timer

(2) C-Language Format

```
ER_RET timer_start( uint8_t ch );
```

(3) Parameter

I/O	Parameter	Description
I	uint8_t ch	Channel selection argument 0: channel- 0 1: channel- 1 2: channel- 2 3: channel- 3

(4) Function

This function starts the timer selected by the channel selection argument.
A parameter error is returned if the selected channel is not 0 to 3.

(5) Return Value

Return Value	Meaning
ER_OK	Timer started
ER_PARAM	Parameter error - The selected channel is NOT 0 to 3

6.2.5 Stopping Timer

timer_stop

(1) Description

Stopping the timer

(2) C-Language Format

```
ER_RET timer_stop( uint8_t ch );
```

(3) Parameter

I/O	Parameter	Description
I	uint8_t ch	Channel selection argument 0: channel- 0 1: channel- 1 2: channel- 2 3: channel- 3

(4) Function

This function stops the timer selected by the channel selection argument.

(5) Return Value

Return Value	Meaning
ER_OK	Timer stopped
ER_PARAM	Parameter error - The selected channel is NOT 0 to 3

6.2.6 Checking the Timer Activation

timer_check_act

(1) Description

Checking the timer activation

(2) C-Language Format

```
ER_RET timer_check_act( uint8_t ch );
```

(3) Parameter

I/O	Parameter	Description
I	uint8_t ch	Channel selection argument 0: channel- 0 1: channel- 1 2: channel- 2 3: channel- 3

(4) Function

This function checks whether the timer selected by the channel selection argument is active or stopped.

(5) Return Value

Return Value	Meaning
1	The selected timer is active
0	The selected timer is stopped
ER_PARAM	Parameter error - The selected channel is NOT 0 to 3

6.3 UART Control

6.3.1 Initialization of UART Module

uart_init

(1) Description

Initialization of the UART module

(2) C-Language Format

```
ER_RET uart_init(uint8_t ch);
```

(3) Parameter

I/O	Parameter		Description
I	uint8_t	ch	Channel selection argument 0: channel-0 1: channel-1

(4) Function

This function initializes several parameters such as the baud rate and bit size for the channel selected by the channel selection argument.
ER_PARAM (parameter error) is returned if the selected channel is not 0 or 1.
Selection of the channel is defined in system_RIN32M3.h.

(5) Return Value

Return Value	Meaning
ER_OK	Initialization succeeded
ER_PARAM	Parameter error - The selected channel is not 0 or 1

6.3.2 Transmission of One Byte of Character Data through UART

uart_write

(1) Description

Transmission of one byte of character data through UART

(2) C-Language Format

```
ER_RET uart_write(uint8_t ch,uint8_t data);
```

(3) Parameter

I/O	Parameter		Description
I	uint8_t	ch	Channel selection argument 0: channel-0 1: channel-1
I	uint8_t	data	One byte of character data for transmission

(4) Function

This function transmits one byte of character data to the selected channel. However, if TX_FIFO is full, the transmission is halted until TX_FIFO becomes empty.
ER_PARAM is returned if the selected channel is not 0 or 1.
Selection of the channel is defined in system_RIN32M3.h.

(5) Return Value

Return Value	Meaning
ER_OK	Transmission succeeded
ER_PARAM	Parameter error - The selected channel is not 0 or 1

6.3.3 Reception of One Byte of Character Data through UART

uart_read

(1) Description

Reception of one byte of character data through UART

(2) C-Language Format

```
ER_RET uart_read(uint8_t ch,uint8_t *data);
```

(3) Parameter

I/O	Parameter	Description
I	uint8_t ch	Channel selection argument 0: channel-0 1: channel-1
O	uint8_t * data	Pointer to the received byte of character data

(4) Function

This function receives one byte of character data from the selected channel. If the buffer holds received data, it is passed as the argument that points to the data and the return value is 1. The return value is 0 if the buffer has no received data. ER_PARAM is returned if the selected channel is other than 0 or 1.
Selection of the channel is defined in system_RIN32M3.h.

(5) Return Value

Return Value	Meaning
1	The buffer holds received data
0	No received data
ER_PARAM	Parameter error - The selected channel is not 0 or 1

6.3.4 Confirming Presence of Received Data

uart_check_receivedata

(1) Description

Checking the presence of received data

(2) C-Language Format

```
ER_RET uart_check_receivedata(uint8_t ch);
```

(3) Parameter

I/O	Parameter	Description
I	uint8_t ch	Channel selection argument 0: channel-0 1: channel-1

(4) Function

This function checks whether RX_FIFO of the selected channel is empty.
ER_PARAM is returned if the selected channel is not 0 or 1.
Selection of the channel is defined in system_RIN32M3.h.

(5) Return Value

Return Value	Meaning
1	The buffer holds received data
0	No received data
ER_PARAM	Parameter error - The selected channel is not 0 or 1

6.4 IIC Control

6.4.1 Initialization of IIC Controller

iic_init

(1) Description

Initialization of the IIC controller

(2) C-Language Format

```
ER_RET iic_init(uint8_t ch);
```

(3) Parameter

I/O	Parameter	Description
I	uint8_t ch	Channel selection argument 0: channel-0 1: channel-1

(4) Function <R>

This function makes initial settings for the IIC controller of the selected channel.
ER_PARAM is returned if the selected channel is not 0 or 1.

- IIC clock setting
  - > Fast mode : 400 kHz
- IIC timing setting
  - > Stop and start interval : 130 × PCLK cycle (ns)
    SCL low-level width : 130 × PCLK cycle (ns)
    SCL high-level width : 116 × PCLK cycle (ns)
  - > Setup cycles
    Start condition : 116 × PCLK cycle (ns)
    Stop condition : 116 × PCLK cycle (ns)
  - > Hold cycles
    Start condition : 116 × PCLK cycle (ns)
    Data : 32 × PCLK cycle (ns)

Remarks 1. The IIC clock setting "400 kHz" is based on the assumption that both the rise and fall times of SDAn and SCLn are 20 ns. Change the register settings appropriately according to your usage environment. For details, refer to the R-IN32M3 Series User's Manual: Peripheral Modules.

2. PCLK cycle = 10 ns <R>

(5) Return Value

Return Value	Meaning
ER_OK	Initialization succeeded
ER_PARAM	Parameter error - The selected channel is not 0 or 1

6.4.2 Transmission of Start Condition

iic_start_condition

(1) Description

Transmission of a start condition

(2) C-Language Format

```
ER_RET iic_start_condition(uint8_t ch);
```

(3) Parameter

I/O	Parameter	Description
I	uint8_t ch	Channel selection argument 0: channel-0 1: channel-1

(4) Function

This function transmits a start condition to the selected channel.
ER_PARAM is returned if the selected channel is not 0 or 1.

(5) Return Value

Return Value	Meaning
ER_OK	Transmission succeeded
ER_PARAM	Parameter error - The selected channel is not 0 or 1

6.4.3 Transmission of Stop Condition

iic_stop_condition

(1) Description

Transmission of a stop condition

(2) C-Language Format

```
ER_RET iic_stop_condition(uint8_t ch);
```

(3) Parameter

I/O	Parameter	Description
I	uint8_t ch	Channel selection argument 0: channel-0 1: channel-1

(4) Function

This function transmits a stop condition to the selected channel.
ER_PARAM is returned if the selected channel is not 0 or 1.

(5) Return Value

Return Value	Meaning
ER_OK	Transmission succeeded
ER_PARAM	Parameter error - The selected channel is not 0 or 1

6.4.4 Transmission of One Byte of Character Data

iic_write

(1) Description

Transmission of one byte of character data

(2) C-Language Format

```
ER_RET iic_write(uint8_t ch, uint8_t data);
```

(3) Parameter

I/O	Parameter		Description
I	uint8_t	ch	Channel selection argument 0: channel-0 1: channel-1
I	uint8_t	data	One byte of character data for transmission

(4) Function

This function transmits one byte of character data to the selected channel.
ER_PARAM is returned if the selected channel is not 0 or 1.
ER_NG (transmission failed) is returned in cases where ACK is not returned from the device each time 8-bit data is transmitted.

(5) Return Value

Return Value	Meaning
ER_OK	Transmission succeeded
ER_NG	Transmission failed - ACK is NOT returned from the device
ER_PARAM	Parameter error - The selected channel is not 0 or 1

6.4.5 Reception of One Byte of Character Data

iic_read

(1) Description

Reception of one byte of character data

(2) C-Language Format

```
ER_RET iic_read(uint8_t ch, uint8_t *data, uint32_t last);
```

(3) Parameter

I/O	Parameter		Description
I	uint8_t	ch	Channel selection argument 0: channel-0 1: channel-1
O	uint8_t *	data	Pointer to the received byte of character data
I	uint32_t	last	Last data designation argument 0: Data other than the last data Others: Last data

(4) Function

This function receives one byte of character data from the selected channel.
ER_PARAM is returned if the selected channel is not 0 or 1.
If the last data designation argument is 0, ACK is output; otherwise, ACK is NOT output.

(5) Return Value

Return Value	Meaning
ER_OK	Reception succeeded
ER_PARAM	Parameter error - The selected channel is not 0 or 1

6.5 CSI Control

6.5.1 Initialization of CSI Controller

csi_init

(1) Description

Initialization of the CSI controller

(2) C-Language Format

```
ER_RET csi_init(uint32_t ch, uint32_t mode);
```

(3) Parameter

I/O	Parameter	Description
I	uint32_t ch	Channel selection argument 0: channel-0 1: channel-1
I	uint32_t mode	Master/slave mode selection argument 0: master 1: slave

(4) Function

This function makes initial settings for the CSI controller selected by the channel selection argument.
ER_PARAM is returned if the channel selection argument or master/slave selection argument is not 0 or 1.

Initial settings common to master and slave modes are as follows.

- CSI data setting : Data length is 8, MSB first, no error detection
- CSI timing setting
  - Setup cycle : 0.5 serial clock cycles
  - Hold cycle : 0.5 serial clock cycles

Initial settings for each selected mode are as follows.

(a) When master mode is selected

Serial clock frequency: 16.667 MHz
Default level: High

(b) When slave mode is selected

CSI clock setting: Use the input clock from master

Remarks 1. The chip select pin is NOT used.
2. FIFO is NOT used.

(5) Return Value

Return Value	Meaning
ER_OK	Initialization succeeded
ER_PARAM	Parameter error - The specified channel or mode is not 0 or 1

6.5.2 Transmission of One Byte of Character Data

csi_write

(1) Description

Transmission of one byte of character data

(2) C-Language Format

```
ER_RET csi_write(uint32_t ch, uint8_t data);
```

(3) Parameter

I/O	Parameter	Description
I	uint32_t ch	Channel selection argument 0: channel-0 1: channel-1
I	uint8_t data	One byte of character data for transmission

(4) Function

This function transmits one byte of character data when the selected channel is in Tx mode. If the CSI controller is not in Tx mode while in master mode, it is placed in Tx mode.
ER_PARAM is returned if the channel selection argument or master/slave mode selection argument is not 0 or 1.
ER_INVAL (mode error) is returned if the CSI controller is not in Tx mode while in slave mode.

(5) Return Value

Return Value	Meaning
ER_OK	Transmission succeeded
ER_PARAM	Parameter error - The selected channel is not 0 or 1
ER_INVAL	Mode error - The CSI controller is not in Tx mode while in slave mode

6.5.3 Reception of One Byte of Character Data

csi_read

(1) Description

Reception of one byte of character data

(2) C-Language Format

```
ER_RET csi_read(uint32_t ch, uint8_t* data);
```

(3) Parameter

I/O	Parameter	Description
I	uint32_t ch	Channel selection argument 0: channel-0 1: channel-1
O	uint8_t * data	Pointer to the received byte of character data

(4) Function

This function receives one byte of character data when the selected channel is in Rx mode. If the CSI controller is not in Rx mode while in master mode, it is placed in Rx mode.
ER_PARAM is returned if the channel selection argument or master/slave mode selection argument is not 0 or 1, and ER_INVAL (mode error) is returned if the CSI controller is not in Rx mode.
ER_INVAL is also returned if the CSI controller is not in Rx mode while in slave mode.

(5) Return Value

Return Value	Meaning
ER_OK	Reception succeeded
ER_PARAM	Parameter error - The selected channel is not 0 or 1
ER_INVAL	Mode error - The CSI controller is not in Rx mode

6.5.4 Confirmation of Transmission Data (for Slave)

csi_check_tx

(1) Description

Confirming data for transmission (for slave)

(2) C-Language Format

```
ER_RET csi_check_tx(uint32_t ch);
```

(3) Parameter

I/O	Parameter	Description
I	uint32_t ch	Channel selection argument 0: channel-0 1: channel-1

(4) Function

When the selected channel is in Tx mode, the return value is presence of CSI transmission data.
When the CSI controller is in master mode, ER_OK (no transmission data) is always returned because transmission data is not stored.
ER_PARAM is returned if the channel selection argument is not 0 or 1.
If the CSI controller is not in Tx mode, ER_INVAL (mode error) is returned.

(5) Return Value

Return Value	Meaning
ER_OK	No transmission data
ER_NOTYET	Transmission data is present
ER_PARAM	Parameter error - The selected channel is not 0 or 1
ER_INVAL	Mode error - The CSI controller is not in Tx mode

6.5.5 Confirmation of Received Data (for Slave)

csi_check_rx

(1) Description

Confirming received data (for slave)

(2) C-Language Format

```
ER_RET csi_check_rx(uint32_t ch);
```

(3) Parameter

I/O	Parameter	Description
I	uint32_t ch	Channel selection argument 0: channel-0 1: channel-1

(4) Function

When the selected channel is in Rx mode, the return value is presence of CSI received data.
When the CSI controller is in master mode, ER_ NOTYET (no received data) is always returned because received data is not stored.
ER_PARAM is returned if the channel selection argument is not 0 or 1.
If the CSI controller is not in Rx<R> mode, ER_INVAL (mode error) is returned.

(5) Return Value

Return Value	Meaning
ER_OK	Received data is present
ER_NOTYET	No received data
ER_PARAM	Parameter error - The selected channel is not 0 or 1
ER_INVAL	Mode error - The CSI controller is not in Rx mode

6.5.6 Switching Tx/Rx Mode (for Slave)

csi_change_mode

(1) Description

Switching Tx/Rx mode

(2) C-Language Format

```
ER_RET csi_change_mode(uint32_t ch, uint32_t mode);
```

(3) Parameter

I/O	Parameter	Description
I	uint32_t ch	Channel selection argument 0: channel-0 1: channel-1
I	uint32_t mode	Transfer mode selection argument 0: Rx mode 1: Tx mode

(4) Function

This function sets CSI Tx/Rx mode for the selected channel.
ER_PARAM is returned if the channel selection argument or transfer mode selection argument is not 0 or 1.

- If the transfer mode selection argument is Rx mode, the setting is changed as below.
  - > Stopping Tx operation
  - > Enabling Rx operation
- If the transfer mode selection argument is Tx mode, the setting is changed as below.
  - > Tx operation is permitted
  - > Rx operation is prohibited

(5) Return Value

Return Value	Meaning
ER_OK	Mode switching succeeded
ER_PARAM	Parameter error - The selected channel is not 0 or 1 - The selected mode is not 0 or 1

6.6 DMA Control

6.6.1 Copying Memory (DMA Transfer)

dmac_memcpy

(1) Description

Copying memory (DMA transfer)

(2) C-Language Format

```
void *dmac_memcpy(void *dst, const void *src, uint32_t n);
```

(3) Parameter

I/O	Parameter		Description
I	void*	dst	Destination address
I	void*	src	Source address
I	uint32_t	n	Number of transfer bytes

(4) Function

This function copies memory from the source address to the destination address by DMA transfer.
The dst (destination address) is returned at the end of transfer.

(5) Return Value

Return Value	Meaning
dst	Destination address

6.7 Serial Flash ROM Control

6.7.1 Initialization of SPI Bus Controller

sromc_init

(1) Description

Initialization of the SPI bus controller

(2) C-Language Format

```
void sromc_init(void);
```

(3) Parameter

None

(4) Function

This function initializes the serial flash ROM controller.

- Serial flash ROM clock setting
  - > Serial flash ROM clock frequency : 25 MHz
  - > Serial clock default level : High
- Serial flash ROM read mode setting : Data length is 8, MSB first
- Serial flash ROM timing setting
  - > Data I/O
    - Input setup cycle : 0.5 serial clock cycles
    - Output hold cycle : 0.5 serial clock cycles
  - > Minimum width at high level of the device select signal of the SPI bus : 8 serial clock cycles
  - > Setup cycles
    - Device select signal of the SPI bus : 1.5 serial clock cycles
    - Serial data output : 0.5 serial clock cycles
  - > Hold cycles
    - Device select signal of the SPI bus : 1.5 serial clock cycles
    - Serial data output : 0.5 serial clock cycles

(5) Return Value

None

6.7.2 Writing Data to SPI Bus

sromc_write

(1) Description

Writing data to the SPI bus

(2) C-Language Format

```
void sromc_write(uint8_t data, uint32_t first, uint32_t last);
```

(3) Parameter

I/O	Parameter		Description
I	uint8_t	data	One byte of character data for writing
I	uint32_t	first	Flag for mode setting for direct access to the SPI bus 0: No direct access mode setting Others: Direct access mode setting
I	uint32_t	last	Flag for ROM access mode setting 0: No ROM access mode setting Others: ROM access mode setting

(4) Function

This function writes data given by the "data" argument to the SPI bus.

If the "first" argument is not 0, set it to direct access mode before data is written.

If the "last" argument is not 0, set it to ROM access mode after data is written.

(5) Return Value

None

6.7.3 Reading Data from SPI Bus

sromc_read

(1) Description

Reading data from the SPI bus

(2) C-Language Format

```
void sromc_read(uint8_t* data, uint32_t first, uint32_t last);
```

(3) Parameter

I/O	Parameter		Description
O	uint8_t *	data	Pointer to one byte of character data read
I	uint32_t	first	Flag for mode setting for direct access to the SPI bus 0: No direct access mode setting Others: Direct access mode setting
I	unit32_t	last	Flag for ROM access mode setting 0: No ROM access mode setting Others: ROM access mode setting

(4) Function

This function stores data read form the SPI bus into the pointer specified by the "data" argument.

If the "first" argument is not 0, set it to direct access mode before data is read.

If the "last" argument is not 0, set it to ROM access mode after data is read.

(5) Return Value

None

6.8 Watchdog Timer Control

6.8.1 Initialization of Watchdog Timer

wdt_init

(1) Description

Initialization of the watchdog timer

(2) C-Language Format

```
ER_RET wdt_init(void);
```

(3) Parameter

None

(4) Function

This function initializes the watchdog timer.

- Error mode : Reset mode (reset when the counter overflows)
- Counter overflow interval time : 1.342 s
- Window open period : 100%

(5) Return Value

Return Value	Meaning
ER_OK	Initialization succeeded

6.8.2 Starting Watchdog Timer

wdt_start

(1) Description

Starting the watchdog timer

(2) C-Language Format

```
ER_RET wdt_start(void);
```

(3) Parameter

None

(4) Function

This function starts the watchdog timer. The watchdog timer cannot be stopped once it is started.

(5) Return Value

Return Value	Meaning
ER_OK	Timer started

6.8.3 Counter Clearing

wdt_clear

(1) Description

Clearing the counter

(2) C-Language Format

```
ER_RET wdt_clear(void);
```

(3) Parameter

None

(4) Function

This function clears the counter value of the watchdog timer.

(5) Return Value

Return Value	Meaning
ER_OK	Counter cleared

6.8.4 Waiting for Reset

wdt_wait_reset

(1) Description

Waiting for a reset

(2) C-Language Format

```
void wdt_wait_reset(void);
```

(3) Parameter

None

(4) Function

This is for waiting for the reset signal for the watchdog timer to be output in response to the overflow of the counter.

(5) Return Value

None

6.9 CAN Control <R>

6.9.1 Enabling CAN controller

can_enable

(1) Description

Enabling the CAN controller

(2) C-Language Format

```
ER_RET can_enable(uint8_t ch)
```

(3) Parameter

I/O	Parameter	Description
I	uint8_t ch	Channel selection argument 0: channel- 0 1: channel- 1

(4) Function

This function starts the CAN controller module of the channel selected by the channel selection argument.
ER_PARAM is returned, if the following error occurs.

- The channel selection argument is not 0 or 1.
- The selected channel is not available.

ER_INVAL is returned, if the following error occurs.

- An error occurs when reading the RAM in the message buffer.
- The CAN module has been activated.

ER_BUSY is returned during software resetting.

Caution: Make sure to call this function before calling can_init.

(5) Return Value

Return Value	Meaning
ER_OK	The CAN controller is enabled normally
ER_PARAM	Parameter error
ER_INVAL	Enabling CAN is disabled
ER_BUSY	During software resetting

6.9.2 Initialization of CAN Controller

can_init

(1) Description

Initialization of the CAN controller

(2) C-Language Format

```
ER_RET can_init(uint8_t ch);
```

(3) Parameter

I/O	Parameter	Description
I	uint8_t ch	Channel selection argument 0: channel-0 1: channel-1

(4) Function

This function initializes the CAN controller module of the channel selected by the channel selection argument, according to the CAN configuration table.

ER_PARAM is returned, if the following error occurs.

- The channel selection argument is not 0 or 1.
- The selected channel is not available.

ER_INVAL is returned, if the CAN controller is not in initialization mode.

(5) Return Value

Return Value	Meaning
ER_OK	The CAN controller is initialized normally
ER_PARAM	Parameter error
ER_INVAL	Initiation of CAN is disabled

6.9.2.2 Configuration Table for CAN Controller Initialization

The configuration table contains some of the register information to be set at calling the initialization driver of the CAN controller.

(1) Table for Setting CAN Controller Channel

can_ch_info

For each channel of the CAN controller, the table for initial setting

・C-Language Format

```
const CAN_CHINFO_TypeDef can_ch_info[CAN_CH_NUM]
```

・CAN_CHINFO_TypeDef Structure Member

I/O	Member	Description
I	uint8_t use	Channel enable setting bit7: Channel enable setting 0: Channel disabled 1: Channel enabled bit6 to bit0: Channel number
I	uint16_t FCNnCMIECTL	Interrupt enable setting (multiple choices allowed) Transmission suspending interrupt is enabled: CAN_CMIECTL_SET_TRXABT Wake-up interrupt is enabled: CAN_CMIECTL_SET_WAKUP Arbitration loss and interrupt are enabled: CAN_CMIECTL_SET_ARBLST CAN protocol error interrupt is enabled: CAN_CMIECTL_SET_PRTERR CAN error status interrupt is enabled: CAN_CMIECTL_SET_ERRSTS Message reception complete interrupt to the message buffer is enabled: CAN_CMIECTL_SET_RX Message transmission complete interrupt from the message buffer is enabled: CAN_CMIECTL_SET_TX

(2) Table for Setting CAN Controller Baud Rate

can_bps_info

For each channel of the CAN controller, the table for setting the baud rate

・C-Language Format

```
const CAN_BPSINFO_TypeDef can_bps_info[CAN_CH_NUM]
```

・CAN_BPSINFO_TypeDef Structure Member

I/O	Member		Description
I	uint8_t	FCNnGMCSPRE	System clock setting
I	uint8_t	FCNnCMBRPRS	Bit-rate prescaler setting
I	uint16_t	FCNnCMBTCTL	Bit-rate setting

Reference: Section 19.13, Baud Rate Settings, in the R-IN32M3 Series User's Manual: Peripheral Modules

(3) Table for Setting ID Mask of CAN Controller Message Buffer

can_msg_msk_info

For each channel of the CAN controller, the table for setting the ID mask of the reception message buffer

・C-Language Format

```
const uint32_t can_msg_msk_info[CAN_CH_NUM][CAN_NUM_OF_MASK]
```

・can_msg_msk_info Alignment

I/O	Member	Description
I	uint32_t	ID mask register (FCNnCMMKCTLmW) setting

Reference: Section 19.7.4, Mask Function, in the R-IN32M3 Series User's Manual: Peripheral Modules

(4) Table for Setting CAN Controller Message Buffer

can_msg_info

For each channel of the CAN controller, the table for setting the message buffer

・C-Language Format

```
const CAN_MSGINFO_TypeDef can_msg_info[CAN_CH_NUM][CAN_MSG_BUF_NUM]
```

・CAN_MSGINFO_TypeDef Structure Member

I/O	Member		Description
I	uint32_t	FCNnMmMID0W	CAN_ID setting Standard ID specification: CAN_SET_STD_ID(CAN_ID) Extended ID specification: CAN_SET_EXT_ID(CAN_ID)
I	uint8_t	FCNnMmSTRB	Message buffer mode specification Transmission mode: CAN_MSGBUF_INI_TX Reception mode (no mask register is used): CAN_MSGBUF_INI_RX Reception mode (mask 1 register is used): CAN_MSGBUF_INI_RX_MSK1 Reception mode (mask 2 register is used): CAN_MSGBUF_INI_RX_MSK2 Reception mode (mask 3 register is used): CAN_MSGBUF_INI_RX_MSK3 Reception mode (mask 4 register is used): CAN_MSGBUF_INI_RX_MSK4 Reception mode (mask 5 register is used): CAN_MSGBUF_INI_RX_MSK5 Reception mode (mask 6 register is used): CAN_MSGBUF_INI_RX_MSK6 Reception mode (mask 7 register is used): CAN_MSGBUF_INI_RX_MSK7 Reception mode (mask 8 register is used): CAN_MSGBUF_INI_RX_MSK8
I	uint8_t	FCNnMmDTLGB	DLC (Data Length Code) setting Transmission mode: 0 to 8 Reception mode: 0

6.9.3 Forced Termination of CAN Controller

can_shutdown

(1) Description

Forced termination of the CAN controller

(2) C-Language Format

```
ER_RET can_shutdown(uint8_t ch);
```

(3) Parameter

I/O	Parameter	Description
I	uint8_t ch	Channel selection argument 0: channel-0 1: channel-1

(4) Function

This function forcibly terminates the CAN controller module of the channel selected by the channel selection argument.
ER_PARAM is returned, if the following error occurs.

- The channel selection argument is not 0 or 1.
- The selected channel is not available.

ER_INVAL is returned, if the module is in the active state even after forcibly terminating the CAN controller.

(5) Return Value

Return Value	Meaning
ER_OK	The CAN controller is initialized normally
ER_PARAM	Parameter error
ER_INVAL	Forced termination of CAN is disabled

6.9.4 Acquisition of CAN Operating Mode

can_get_mode

(1) Description

Acquisition of the operating mode for the CAN controller

(2) C-Language Format

```
ER_RET can_get_mode(uint8_t ch);
```

(3) Parameter

I/O	Parameter	Description
I	uint8_t ch	Channel selection argument 0: channel- 0 1: channel- 1

(4) Function

This function acquires the operating mode for the CAN controller module of the channel selected by the channel selection argument.

ER_PARAM is returned, if the following error occurs.

- The channel selection argument is not 0 or 1.
- The selected channel is not available.

(5) Return Value

Return Value	Meaning
bit7 = 0	Acquisition of the operating mode is completed ・bit2 - bit0: Operating mode 000b: Initialization mode 001b: Normal mode 101b: Self-test mode ・bit4 - bit3: Power save mode 00b: Non-power save mode 01b: CAN sleep mode 11b: CAN stop mode ・bit7 - bit5: 0
ER_PARAM	Parameter error

6.9.5 Setting CAN Operating Mode

can_set_mode

(1) Description

Setting the operating mode for the CAN controller

(2) C-Language Format

```
ER_RET can_set_mode(uint8_t ch,uint16_t mode);
```

(3) Parameter

I/O	Parameter	Description
I	uint8_t ch	Channel selection argument 0: channel-0 1: channel-1
I	uint8_t mode	Operating mode argument SET_CAN_INIT: Initialization mode SET_CAN_NORM: Normal mode SET_CAN_SELF: Self-test mode

(4) Function

This function sets the operating mode for the CAN controller module of the channel selected by the channel selection argument.

ER_PARAM is returned, if the following error occurs.

- The channel selection argument is not 0 or 1.
- The selected channel is not available.

ER_INVAL is returned, if the following error occurs.

- The operating mode argument is the initialization mode, though the current operating mode is in initialization mode.
- The operating mode argument is not the initialization mode, though the current operating mode is not in initialization mode.

Caution: When switching from the mode other than the initialization mode to the other operating mode, make sure to set the initialization mode before switching.

(5) Return Value

Return Value	Meaning
ER_OK	The operating mode is set normally
ER_PARAM	Parameter error
ER_INVAL	Setting of the operating mode is disabled

6.9.6 Acquisition of CAN Reception Data (CANID, Data, DLC)

can_get_id_data_dlc

(1) Description

Acquisition of CAN ID, reception data, and DLC from the reception message buffer of the CAN controller

(2) C-Language Format

```
ER_RET can_get_id_data_dlc(uint8_t ch,uint8_t bufno,
                           uint32_t *canid,uint8_t *data,uint8_t *dlc);
```

(3) Parameter

I/O	Parameter		Description
I	uint8_t	ch	Channel selection argument 0: channel-0 1: channel-1
I	uint8_t	bufno	Message buffer number of the reception-data acquisition destination
O	uint32_t	*canid	Pointer to the CAN_ID storage destination
O	uint8_t	*data	Pointer to the reception-data storage destination
O	uint8_t	*dlc	Pointer to the storage destination for the number of reception data

(4) Function

This function acquires the CAN ID, reception data, and DLC from the CAN-controller message buffer, by the specified channel and buffer number.

ER_PARAM is returned, if the following error occurs.

- The channel selection argument is not 0 or 1.
- The selected channel is not available.
- The buffer number is not within the supported range.
- The message buffer is not available.

ER_INVAL is returned, if the following error occurs.

- The message buffer has no new data.
- The message buffer is under updating.

Remark: Acquire the message buffer number from the acquisition driver for the reception buffer number of CAN data.

(5) Return Value

Return Value	Meaning
ER_OK	The reception data is acquired normally
ER_PARAM	Parameter error
ER_INVAL	Acquiring the reception data is disabled

6.9.7 Acquisition of CAN Reception Data (Data, DLC)

can_get_data_dlc

(1) Description

Acquisition of reception data and DLC from the reception message buffer of the CAN controller

(2) C-Language Format

```
ER_RET can_get_data_dlc(uint8_t ch,uint8_t bufno,uint8_t *data,uint8_t *dlc)
```

(3) C-Language Format

I/O	Parameter		Description
I	uint8_t	ch	Channel selection argument 0: channel-0 1: channel-1
I	uint8_t	bufno	Message buffer number of the reception-data acquisition destination
O	uint8_t	*data	Pointer to the reception-data storage destination
O	uint8_t	*dlc	Pointer to the storage destination for the number of reception data

(4) Function

This function acquires the reception data and DLC from the CAN-controller message buffer, by the specified channel and buffer number.

ER_PARAM is returned, if the following error occurs.

- The channel selection argument is not 0 or 1.
- The selected channel is not available.
- The buffer number is not within the supported range.
- The message buffer is not available.

ER_INVAL is returned, if the following error occurs.

- The message buffer has no new data.
- The message buffer is under updating.

Remark: Acquire the message buffer number from the acquisition driver for the reception buffer number of CAN data.

(5) Return Value

Return Value	Meaning
ER_OK	The reception data is acquired normally
ER_PARAM	Parameter error
ER_INVAL	Acquiring the reception data is disabled

6.9.8 Setting CAN Transmission Data (CAN_ID, Data, DLC)

can_set_id_data_dlc

(1) Description

Setting the transmission data to the CAN-controller message buffer, by specifying CAN_ID and DLC

(2) C-Language Format

```
ER_RET can_set_id_data_dlc(uint8_t ch,uint8_t bufno,
                           uint32_t canid,uint8_t *data,uint8_t dlc)
```

(3) Parameter

I/O	Parameter		Description
I	uint8_t	ch	Channel selection argument 0: channel-0 1: channel-1
I	uint8_t	bufno	Message buffer number of the transmission-data set destination
I	uint32_t	canid	CAN_ID Standard ID: CAN_SET_STD_ID(canid) Extended ID: CAN_SET_EXT_ID(canid)
I	uint8_t	*data	Pointer to the transmission data
I	uint8_t	dlc	Size of the transmission data

(4) Function

This function sets the data (CAN_ID, Data, DLC) in the CAN-controller message buffer, by the specified channel and buffer number.

ER_PARAM is returned, if the following error occurs.

- The channel selection argument is not 0 or 1.
- The selected channel is not available.
- The buffer number is not within the supported range.
- The message buffer is not available.

ER_INVAL is returned, if data is being transmitted.

Remark: Set CAN_ID with the CAN_SET_STD_ID or CAN_SET_EXT_ID macro.

(5) Return Value

Return Value	Meaning
ER_OK	The reception data is acquired normally
ER_PARAM	Parameter error
ER_INVAL	Acquiring the transmission data is disabled

6.9.9 Setting CAN Transmission Data

can_set_data

(1) Description

Setting the transmission data to the CAN-controller message buffer

(2) C-Language Format

```
ER_RET can_set_data(uint8_t ch,uint8_t bufno,uint8_t *data)
```

(3) Parameter

I/O	Parameter		Description
I	uint8_t	ch	Channel selection argument 0: channel-0 1: channel-1
I	uint8_t	bufno	Message buffer number of the transmission-data set destination
I	uint8_t	*data	Pointer to the transmission data

(4) Function

This function sets the data in the CAN-controller message buffer, by the specified channel and buffer number.
ER_PARAM is returned, if the following error occurs.

- The channel selection argument is not 0 or 1.
- The selected channel is not available.
- The buffer number is not within the supported range.
- The message buffer is not available.

ER_INVAL is returned, if data is being transmitted.

Caution: The initial setting value is transmitted to CAN_ID and DLC.
CAN_ID and DLC for the respective message buffer can be modified from the initial setting value, by calling the set driver of CAN transmission data (CANID, Data, DLC).

(5) Return Value

Return Value	Meaning
ER_OK	The reception data is acquired normally
ER_PARAM	Parameter error
ER_INVAL	Acquiring the transmission data is disabled

6.9.10 Request of CAN Data Transmission

can_tx_req

(1) Description

Request the CAN controller to transmit data

(2) C-Language Format

```
ER_RET can_tx_req(uint8_t ch,uint8_t bufno)
```

(3) Parameter

I/O	Parameter		Description
I	uint8_t	ch	Channel selection argument 0: channel-0 1: channel-1
I	uint8_t	bufno	Message buffer number of the transmission request

(4) Function

This function requests the CAN-controller message buffer to transmit data, by the specified channel and buffer number.
ER_PARAM is returned, if the following error occurs.

- The channel selection argument is not 0 or 1.
- The selected channel is not available.
- The buffer number is not within the supported range.
- The message buffer is not available.

ER_INVAL is returned, if transmission data is not set.

(5) Return Value

Return Value	Meaning
ER_OK	Requesting the data transmission is enabled
ER_PARAM	Parameter error
ER_INVAL	Requesting the data transmission is disabled

6.9.11 Acquisition of CAN Data Transmission Information

can_get_txinfo

(1) Description

Acquisition of the data transmission information of the CAN controller

(2) C-Language Format

```
ER_RET can_get_txinfo(uint8_t ch,uint8_t bufno)
```

(3) Parameter

I/O	Parameter	Description
I	uint8_t ch	Channel selection argument 0: channel-0 1: channel-1
I	uint8_t bufno	Buffer number

(4) Function

This function acquires the data transmission state of the CAN-controller message buffer, by the specified channel and buffer number.

ER_PARAM is returned, if the following error occurs.

- The channel selection argument is not 0 or 1.
- The selected channel is not available.
- The buffer number is not within the supported range.
- The message buffer is not available.

ER_INVAL is returned, if data is being transmitted.

(5) Return Value

Return Value	Meaning
ER_OK	No transmission data (buffer empty)
ER_PARAM	Parameter error
ER_BUSY	Data is being transmitted

6.9.12 Acquisition of Reception Buffer Number of CAN Data

can_get_rxinfo

(1) Description

Acquisition of the message buffer number for the CAN-controller data reception

(2) C-Language Format

```
ER_RET can_get_rxinfo(uint8_t ch,uint8_t bufno)
```

(3) Parameter

I/O	Parameter	Description
I	uint8_t ch	Channel selection argument 0: channel-0 1: channel-1
I	uint8_t bufno	Buffer number to start the detection of message buffer

(4) Function

This function detects the reception data from the CAN-controller message buffer of the specified channel.
Detecting is started from the buffer number, and continued to perform to the following message buffers.
ER_PARAM is returned, if the following error occurs.

- The channel selection argument is not 0 or 1.
- The selected channel is not available.
- The buffer number is not within the supported range.
- The message buffer is not available.

ER_INVAL is returned, if no reception data is detected.

(5) Return Value

Return Value	Meaning
bit7 = 0	Reception data detected bit6 – bit0: Reception buffer number
ER_PARAM	Parameter error
ER_INVAL	No reception data

6.9.13 Acquisition of CAN Cannel Status

can_get_ch_status

(1) Description

Acquisition of the CAN-controller channel status

(2) C-Language Format

```
ER_RET can_get_ch_status(uint8_t ch)
```

(3) Parameter

I/O	Parameter	Description
I	uint8_t ch	Channel selection argument 0: channel-0 1: channel-1

(4) Function

This function acquires the CAN-controller channel status.
ER_PARAM is returned, if the following error occurs.

- The channel selection argument is not 0 or 1.
- The selected channel is not available.

(5) Return Value

Return Value	Meaning
bit7 = 0	Channel status acquired bit0: Transmission completed from message buffer m bit1: Reception completed to message buffer m bit2: CAN error status bit3: CAN protocol error bit4: Arbitration lost bit5: Return from CAN sleep mode bit6: Transmission suspended
ER_PARAM	Parameter error

6.9.14 Clearing CAN Channel Status

can_clr_ch_status

(1) Description

Clearing the channel status of the CAN controller

(2) C-Language Format

```
ER_RET can_clr_ch_status(uint8_t ch,uint8_t clrdat)
```

(3) Parameter

I/O	Parameter		Description
I	uint8_t	ch	Channel selection argument 0: channel-0 1: channel-1
I	uint8_t	clrdat	Channel status cleared data bit0: Transmission completed from message buffer m bit1: Reception completed to message buffer m bit2: CAN error status bit3: CAN protocol error bit4: Arbitration lost bit5: Return from CAN sleep mode bit6: Transmission suspended

(4) Function

This function clears the channel status of the CAN controller.
ER_PARAM is returned, if the following error occurs.

- The channel selection argument is not 0 or 1.
- The selected channel is not available.

(5) Return Value

Return Value	Meaning
ER_OK	Channel status clearing enabled
ER_PARAM	Parameter error

6.9.15 Acquisition of CAN Bus Status

can_get_bus_staus

(1) Description

Acquisition of the CAN-controller bus status

(2) C-Language Format

```
ER_RET can_get_bus_staus(uint8_t ch)
```

(3) Parameter

I/O	Parameter	Description
I	uint8_t ch	Channel selection argument 0: channel-0 1: channel-1

(4) Function

This function acquires the CAN-controller bus status.
ER_PARAM is returned, if the following error occurs.

- The channel selection argument is not 0 or 1.
- The selected channel is not available.

(5) Return Value

Return Value	Meaning
bit7 = 0	Channel status acquired bit1 - bit0: Status of the reception error counter bit3 - bit2: Status of the transmission error counter bit4: Bus off status bit7 to bit5: 0
ER_PARAM	Parameter error

7. Middleware

Functions of the middleware are explained in this section.

7.1 Lists of Middleware Functions

The API functions in this sample software are listed below.

Table 7.1 EEPROM Functions

Function Name	Description
eep_init	Initialization of EEPROM control
eep_write	Writing EEPROM data
eep_read	Reading EEPROM data

Table 7.2 Parallel Flash ROM Driver Functions

Function Name	Description
flash_init	Initialization of parallel flash ROM control
flash_program	Writing data
flash_read_data	Reading data
flash_erase	Erasing data
flash_read_cfi	Reading CFI data

Table 7.3 Serial Flash ROM Driver Functions

Function Name	Description
sflash_init	Initialization of serial flash ROM control
sflash_program	Programming of data to serial flash ROM
sflash_read	Reading serial flash ROM data
sflash_erase	Erasing serial flash ROM data

7.2 EEPROM Control

7.2.1 Initialization of EEPROM Controller

eep_init

(1) Description

Initialization of the EEPROM controller

(2) C-Language Format

```
ER_RET eep_init(uint8_t ch, uint8_t iic_adr);
```

(3) Parameter

I/O	Parameter	Description
I	uint8_t ch	IIC channel selection argument 0: channel-0 1: channel-1
I	uint8_t iic_adr	Device selection argument (0 to 7)

(4) Function

This function initializes the IIC controller specified by the IIC channel selection argument with iic_init.
The IIC channel selection argument is used as the channel selection argument (iic_ch_eep) of iic_init.
Select the device by the device selection argument for operation on the write cycle or read cycle.
ER_PARAM is returned with iic_init if the IIC channel selection argument is out of range
ER_PARAM is also returned if the device selection argument is out of range

(5) Return Value

Return Value	Meaning
ER_PARAM	Parameter error - The channel selection argument is NOT 0 or 1 - The device selection argument is out of range
ER_OK	Initialization succeeded

7.2.2 Writing EEPROM Data

eep_write

(1) Description

Writing EEPROM data

(2) C-Language Format

```
ER_RET eep_write(uint32_t eep_addr, uint8_t *data, uint32_t len);
```

(3) Parameter

I/O	Parameter		Description
I	uint32_t	eep_addr	Address where writing of EEPROM data starts
I	uint8_t *	data	Pointer to the data for transmission
I	uint32_t	len	Amount of data to be written

(4) Function

This function activates the IIC controller to transmit the amount of data specified by len from the address specified by eep_adr to the EEPROM. If the amount of data specified by len is over the page size (device-specific), the page size of data is written repeatedly until it reaches the size specified by len. The amount of data specified by len is written from the address specified by *data to the EEPROM
ER_PARAM is returned if the specified end address is over the device capacity (device-specific).

If the value returned from the IIC controller is not ER_OK (transmission succeeded), the operation is as follows.

- Transmission of device selection information : A stop condition is issued and the write failure status is returned.
- Transmission of byte address (high) : A stop condition is issued and the write failure status is returned.
- Transmission of byte address (low) : A stop condition is issued and the write failure status is returned.
- Transmission of the specified amount of data to be written : A stop condition is issued and the write failure status is returned.

(5) Return Value

Return Value	Meaning
ER_OK	Writing succeeded
ER_NG	Writing failed - The result of IIC driver processing is an error.
ER_PARAM	Parameter error - The specified end address is over the device capacity.

7.2.3 Reading EEPROM Data

eep_read

(1) Description

Reading EEPROM data

(2) C-Language Format

```
ER_RET eep_read(uint32_t eep_addr, uint8_t *data, uint32_t len);
```

(3) Parameter

I/O	Parameter		Description
I	uint32_t	eep_addr	Address where reading of data starts
O	uint8_t *	data	Pointer to the received data
I	uint32_t	len	Amount of data to be read

(4) Function

This function activates the IIC controller to receive the amount of data specified by len in the EEPROM from the eep_adr address. The read data in the EEPROM is written from the *data specified address to the len specified size of area. ER_PARAM is returned if the specified end address is over the device capacity (device-specific).

If the value returned from the IIC controller is not ER_OK (transmission succeeded), the operation is as follows.

- Transmission of device selection information : A stop condition is issued and the write failure status is returned.
- Transmission of byte address (high) : A stop condition is issued and the write failure status is returned.

  Transmission of byte address (low) : A stop condition is issued and the write failure status is returned.
- Transmission of device selection information to be read : The communication failure status is returned
- Transmission of the specified amount of data to be written : A stop condition is issued and the write failure status is returned.

(5) Return Value

Return Value	Meaning
ER_OK	Success in reading
ER_NG	Failure in reading - The result of IIC driver processing is an error.
ER_PARAM	Parameter error - The specified end address is over the device capacity.

7.3 Parallel Flash ROM Control

7.3.1 Initialization of Parallel Flash ROM Controller

flash_init

(1) Description

Initialization of the parallel flash ROM controller

(2) C-Language Format

```
ER_RET flash_init(void);
```

(3) Parameter

None

(4) Function

This function initializes the parallel flash ROM controller.

(5) Return Value

Return Value	Meaning
ER_OK	Initialization succeeded

7.3.2 Writing Data

flash_program

(1) Description

Writing data

(2) C-Language Format

```
ER_RET flash_program(uint16_t* buf, uint32_t addr, uint32_t size);
```

(3) Parameter

I/O	Parameter		Description
I	uint16_t*	buf	Address where the write data storage area starts
I	uint32_t	addr	Write address
I	uint32_t	size	Amount of data to be written (in bytes)

(4) Function

The function writes the amount of data specified by the size argument from the write address specified by the addr argument to the parallel flash ROM area.
The written data of the address area specified by the *buf argument is used.

(5) Return Value

Return Value	Meaning
ER_OK	Success
ER_PARAM	Parameter error - The addr argument is NOT a 16-bit address threshold value - The size argument is NOT a 16 bit address threshold value - The additional value of the addr argument and the size argument is over the maximum size of parallel flash ROM

7.3.3 Reading Data

flash_read_data

(1) Description

Reading data

(2) C-Language Format

```
ER_RET flash_read_data(uint16_t* buf, uint32_t addr, uint32_t size);
```

(3) Parameter

I/O	Parameter		Description
O	uint16_t*	buf	Address where writing of read data starts
I	uint32_t	addr	Read address
I	uint32_t	size	Amount of data to be read (in bytes)

(4) Function

The function reads the amount of data specified by the size argument in the parallel flash ROM area from the read address specified by the addr argument.
The data read is written to the address area specified by the *buf argument.

(5) Return Value

Return Value	Meaning
ER_OK	Success
ER_PARAM	Parameter error - The addr argument is NOT a 16-bit address threshold value - The size argument is NOT a 16-bit address threshold value - The additional value of the addr argument and the size argument is over the maximum size of parallel flash ROM

7.3.4 Erasing Data

flash_erase

(1) Description

Erasing data

(2) C-Language Format

```
ER_RET flash_erase(uint32_t addr, uint32_t size);
```

(3) Parameter

I/O	Parameter		Description
I	uint32_t	addr	Erase address
I	uint32_t	size	Amount of data to be erased (in bytes)

(4) Function

The function erases the amount of data specified by the size argument in the parallel flash ROM area from the erase address specified by the addr argument.
The unit of erasure depends on the sector size of parallel flash ROM.

(5) Return Value

Return Value	Meaning
ER_OK	Success
ER_PARAM	Parameter error - The additional value of the addr argument and the size argument is over the maximum size of parallel flash ROM

7.3.5 Reading CFI Data

flash_read_cfi

(1) Description

Reading CFI data

(2) C-Language Format

```
ER_RET flash_read_cfi(uint16_t* buf, uint32_t addr);
```

(3) Parameter

I/O	Parameter		Description
O	uint16_t*	buf	Pointer to the buffer address
I	uint32_t	addr	Read address specification argument

(4) Function

This function refers to CFI data from the address specified by the read address designation argument and stores it into the pointer to the buffer address.

(5) Return Value

Return Value	Meaning
ER_OK	Success
ER_PARAM	Parameter error - The addr argument is NOT a 16-bit address threshold value - The addr argument is over the maximum size of parallel flash ROM
ER_INVAL	CFI is not supported

7.4 Serial Flash ROM Control

7.4.1 Initialization of Serial Flash ROM Controller

sflash_init

(1) Description

Initialization of the serial flash ROM controller

(2) C-Language Format

```
ER_RET sflash_init(uint32_t ch);
```

(3) Parameter

I/O	Parameter	Description
I	uint32_t ch	Driver selection argument 0: Serial flash ROM controller

(4) Function

This function initializes the serial flash ROM controller by using the driver specified by the driver selection argument. ER_PARAM is returned if the driver selection argument is not 0.

(5) Return Value

Return Value	Meaning
ER_PARAM	Parameter error - The driver selection argument is NOT 0.

7.4.2 Programming Data to Serial Flash ROM

sflash_program

(1) Description

Programming data to serial flash ROM

(2) C-Language Format

```
ER_RET sflash_program(uint32_t ch, uint8_t* buf, uint32_t addr, uint32_t size);
```

(3) Parameter

I/O	Parameter		Description
I	uint32_t	ch	Driver selection argument 0: Serial flash ROM controller
I	uint8_t *	buf	Pointer to the write data storage buffer
I	uint32_t	addr	Address where writing starts
I	uint32_t	size	Amount of data to be written

(4) Function

This function writes the buffer data to the serial flash ROM area specified by the addr argument and the size argument by using the driver specified by the driver selection argument.
ER_PARAM is returned if the driver selection argument is not 0.

(5) Return Value

Return Value	Meaning
ER_OK	Success
ER_PARAM	Parameter error - The driver selection argument is NOT 0.

7.4.3 Reading Data from Serial Flash ROM

sflash_read

(1) Description

Reading data from serial flash ROM

(2) C-Language Format

```
ER_RET sflash_read(uint32_t ch, uint8_t* buf, uint32_t addr, uint32_t size);
```

(3) Parameter

I/O	Parameter		Description
I	uint32_t	ch	Driver selection argument 0: Serial flash ROM controller
O	uint8_t *	buf	Pointer to the read data storage buffer
I	uint32_t	addr	Address where reading starts
I	uint32_t	size	Amount of data to be read

(4) Function

This function stores the data of the serial flash ROM area specified by the addr argument and size argument in the buffer by using the driver specified by the driver selection argument.
ER_PARAM is returned if the driver selection argument is not 0.

(5) Return Value

Return Value	Meaning
ER_OK	Success
ER_PARAM	Parameter error - The driver selection argument is NOT 0.

7.4.4 Erasing Serial Flash ROM Data

sflash_erase

(1) Description

Erasing serial flash ROM data

(2) C-Language Format

```
ER_RET sflash_erase(uint32_t ch, uint32_t addr, uint32_t size);
```

(3) Parameter

I/O	Parameter		Description
I	uint32_t	ch	Driver selection argument 0: Serial flash ROM controller
I	uint32_t	addr	Address where erasure starts
I	uint32_t	size	Amount of data to be erased

(4) Function

This function erases the minimum erasable amount of data within the serial flash ROM area as specified by the addr argument and the size argument by using the driver specified by the driver selection argument.
ER_PARAM is returned if the driver selection argument is not 0.

(5) Return Value

Return Value	Meaning
ER_OK	Success
ER_PARAM	Parameter error - The driver selection argument is NOT 0.

8. Example of Application

8.1 OS-less Sample

The operation of the OS-less sample is explained in this section.
The OS-less sample program is a single-task program which does NOT use the hardware OS.

8.1.1 Flow of OS-less Sample

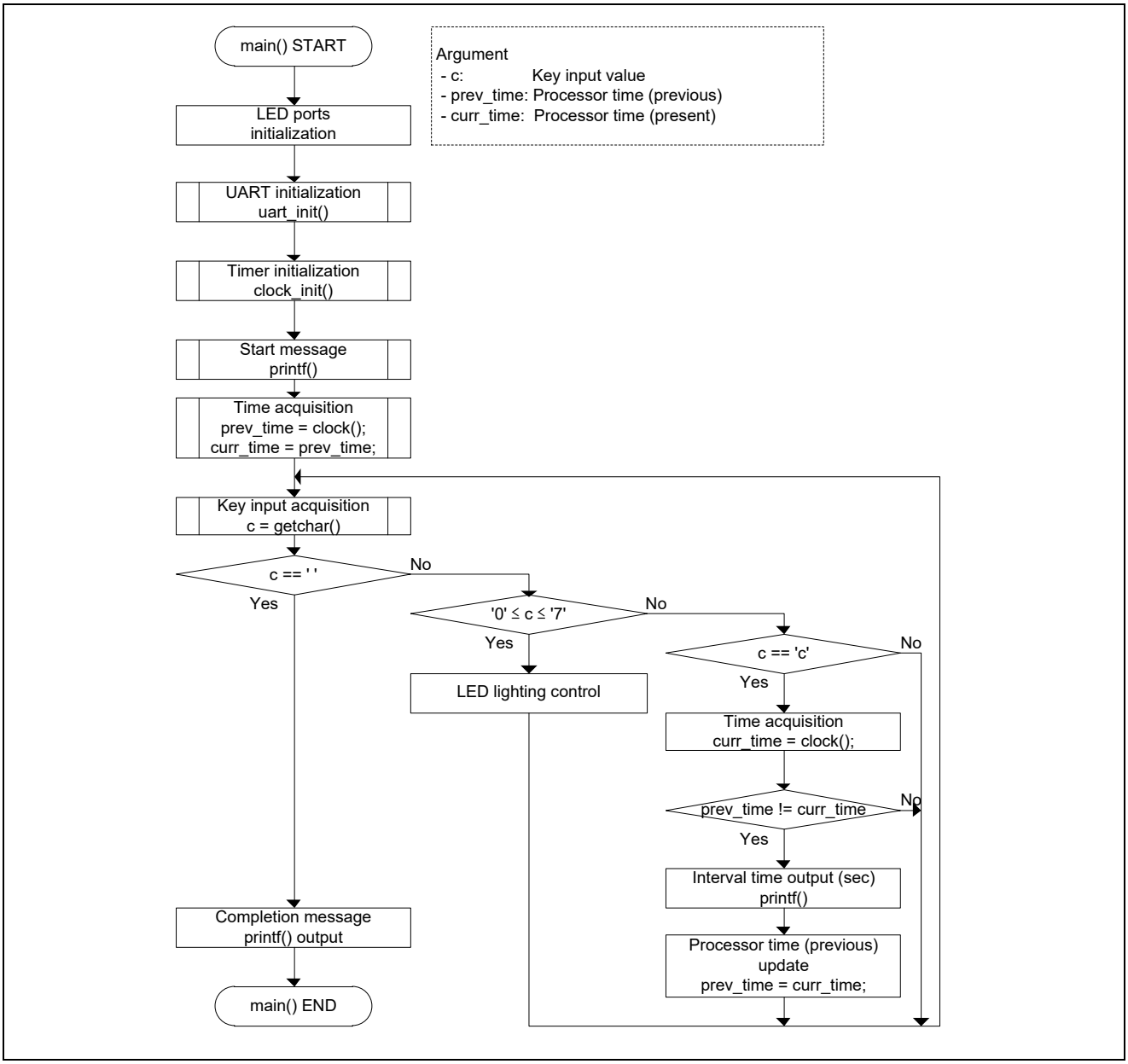

Figure 8.1 Flow Chart of OS-less Sample

8.1.2 Result of Execution

The results of executing the OS-less sample are as follows.

List 8.1 OS-less Sample: Starting

```
hello world
- compiler  =ARM 4.2 (EDG gcc mode)
- boot mode =Parallel Flash
```

Remarks 1. The indication of "compiler" depends on the build environment.
2. The indication of "boot mode" depends on the boot setting

List 8.2 OS-less Sample: LED Control (key input = ’0’ to ’7’)

```
hello world
- compiler  =ARM 4.2 (EDG gcc mode)
- boot mode =Parallel Flash
0                                   (← The level of LED0 is inverted 1 time)
11                                  (← The level of LED1 is inverted 2 times)
222                                 (← The level of LED2 is inverted 3 times)
3333                                (← The level of LED3 is inverted 4 times)
44444                               (← The level of LED4 is inverted 5 times)
555555                              (← The level of LED5 is inverted 6 times)
6666666                             (← The level of LED6 is inverted 7 times)
77777777                            (← The level of LED7 is inverted 8 times)
```

List 8.3 OS-less Sample: indication of interval time (key input = ’c’)

```
hello world
- compiler  =ARM 4.2 (EDG gcc mode)
- boot mode =Parallel Flash
c (interval =464369 ms)             (← indicate time from program start to 1st 'C' key input)
c (interval =2771 ms)               (← Indicate time from 1st 'C' key input to 2nd 'C' key input)
c (interval =187253 ms)             (← indicate time from 2nd 'C' key input to 3rd 'C' key input)
```

List 8.4 OS-less Sample: Finish (key input = ’ ’)

```
hello world
- compiler  =ARM 4.2 (EDG gcc mode)
- boot mode =Parallel Flash
                                    (← break from main process to input ' '.)
bye
```

List 8.5 OS-less Sample: Indication of Interval Time (key input = others)

```
hello world
- compiler  =ARM 4.2 (EDG gcc mode)
- boot mode =Parallel Flash
890-^¥!"#$%&'()=~|qwertyuiop@[QWERTYUIO`{       (← no processing will be taken)
asdfghjkl;:]ASDFGHJKL+*}zxvbnm,./¥ZXVBNM<>?     (← no processing will be taken)
```

8.2 EEPROM Sample

The sample of EEPROM is shown below. The EEPROM sample writes, reads, and downloads data to EEPROM.

Caution: This application expects carriage return/line feed (CRLF) as the end-of-line character.

8.2.1 Flow of EEPROM Sample

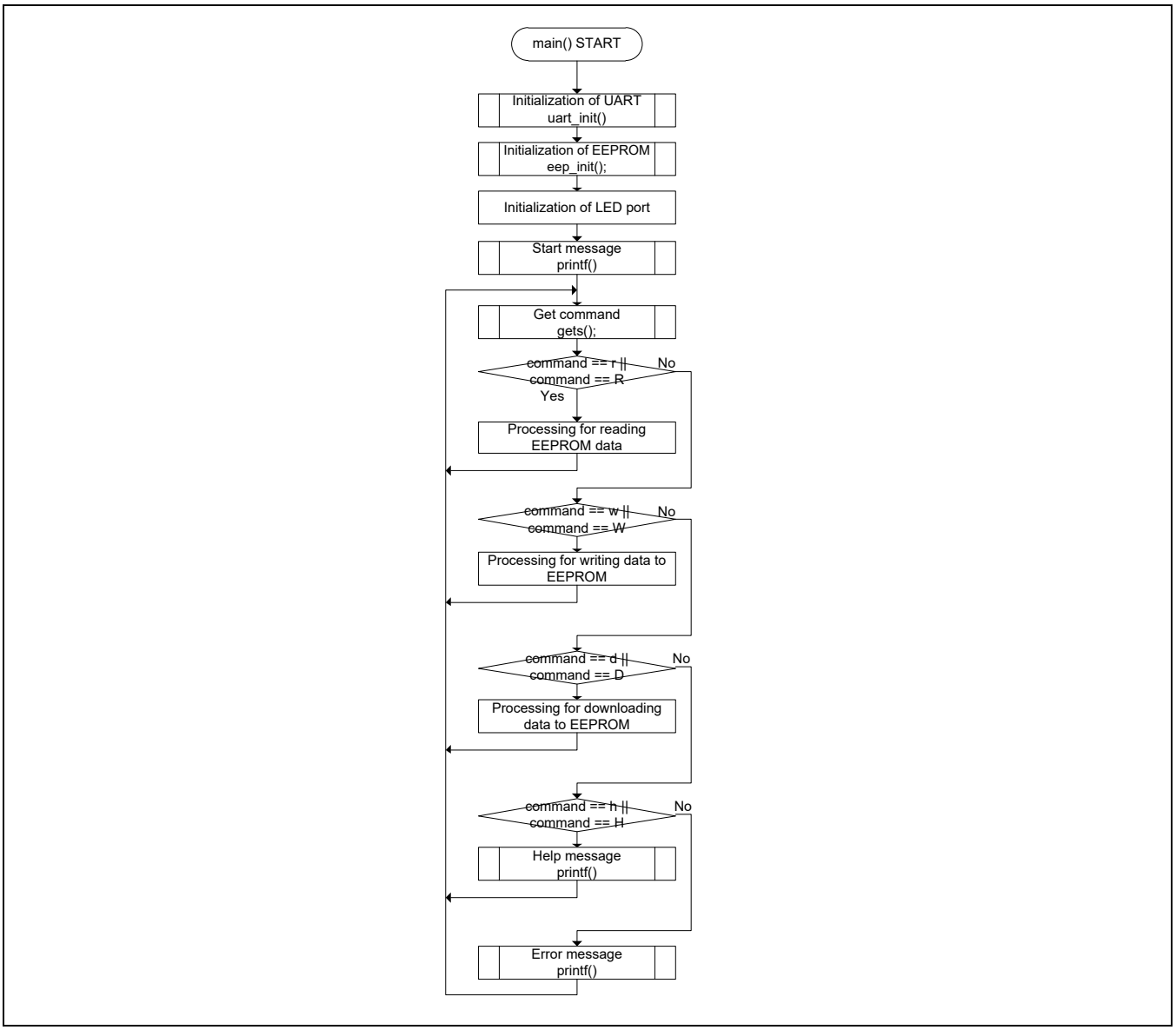

Figure 8.2 Flow Chart of EEPROM Sample (entire flow)

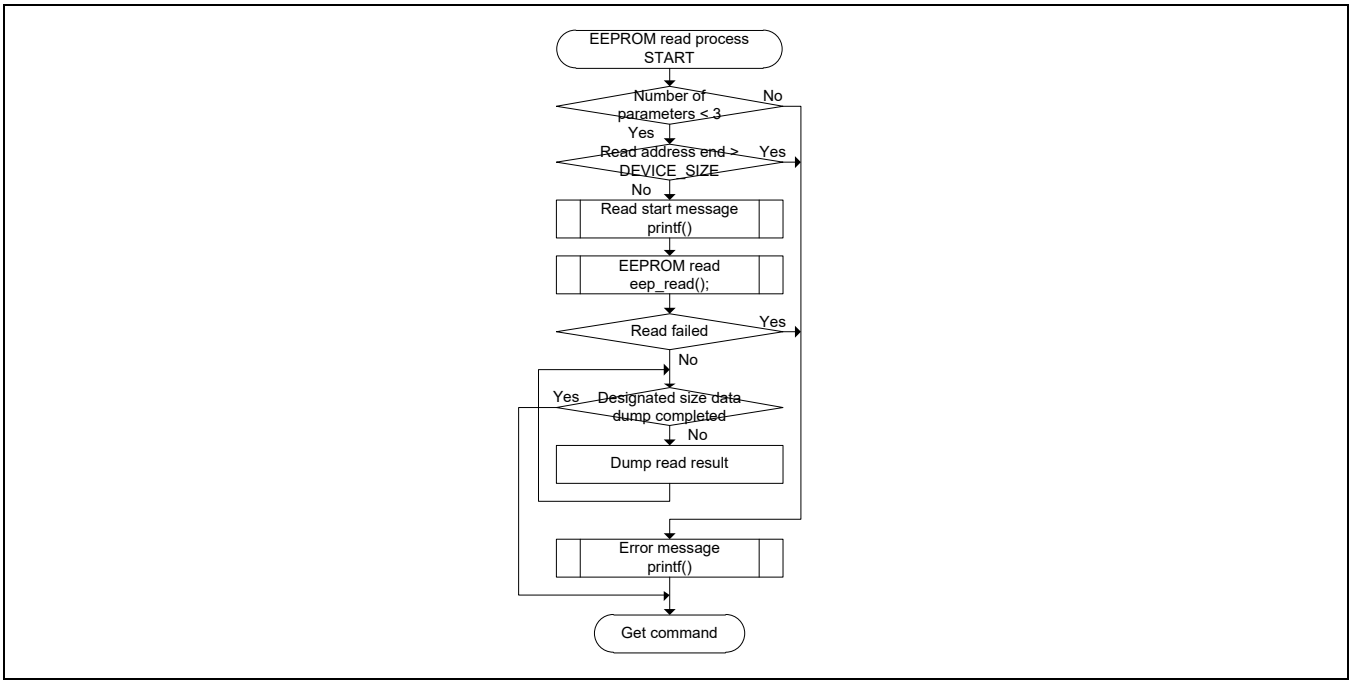

Figure 8.3 Flow Chart of EEPROM Sample (read command executed)

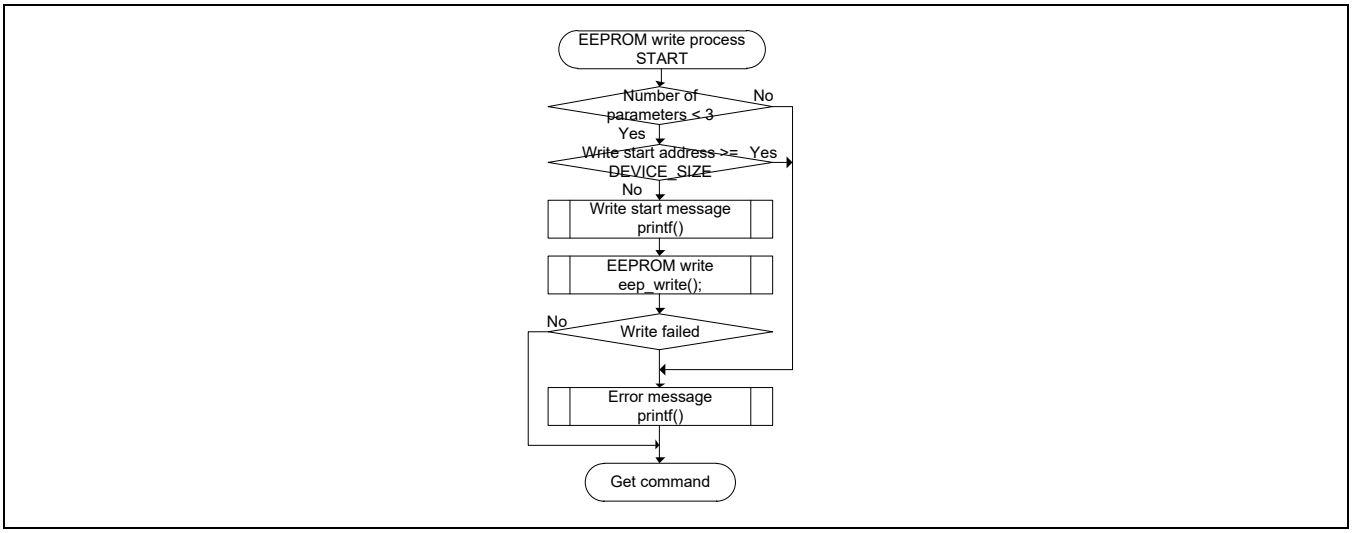

Figure 8.4 Flow Chart of EEPROM Sample (write command executed)

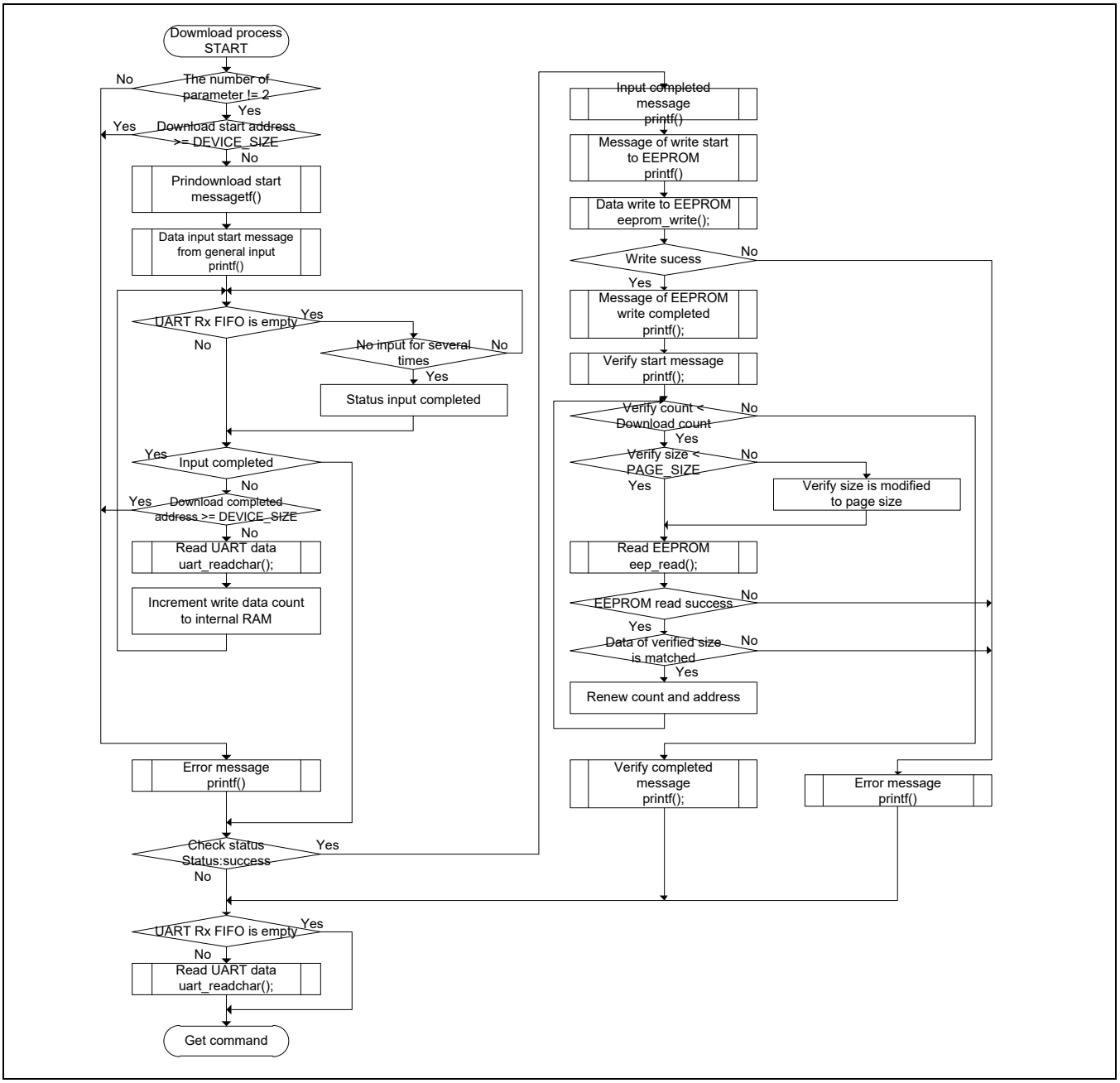

Figure 8.5 Flow Chart of EEPROM Sample (download command executed)

8.2.2 Result of Execution

The results of executing the EEPROM sample are as follows. The applicable device for this result is AT24C16C.

List 8.6 EEPROM Sample: Starting

```
I2C EEPROM Writer
>
```

List 8.7 EEPROM Sample: Read Command Execution

```
I2C EEPROM Writer
> r 0 800
  READ : address=0x00000000 size=0x00000800                  (← read 2 Kbyte data from address 0)
  0x00000000: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  (← indicate every max 16 bytes for 1 line)
  0x00000010: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
  0x00000020: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
                            :
                            :
  0x000007D0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
  0x000007E0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
  0x000007F0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
```

List 8.8 EEPROM Sample: Read Command Parameter Error

```
I2C EEPROM Writer
> r 7ff                                  (← argument needs three or more parameters)
  parameter error : (2) - - -            (← indicate the number of parameters)
> r 7ff 800                              (← max device capacity is 2 Kbytes)
  parameter error : (3) r 7FF 800        (← when last read address is over max capacity)
```

List 8.9 EEPROM Sample: Write Command Execution

```
I2C EEPROM Writer
> w 0 a                                  (← write data 'a' to address '0')
  WRITE : address=0x00000000 data=0x0a   (← indicate a start address and data (ASCII code))
```

List 8.10 EEPROM Sample: Write Command Parameter Error

```
I2C EEPROM Writer
> w 100                                  (← argument needs three or more parameters)
  parameter error : (2) - - -            (← indicate the number of parameters)
> w 8ff a                                (← max device capacity is 2 Kbytes)
  parameter error : (3) w 8ff a          (← when last read address is over max capacity)
```

List 8.11 EEPROM Sample: Download Command Execution

```
I2C EEPROM Writer
> d 0                                                          (← download from address 0)
  DOWNLOAD : address=0x00000000                                (← indicate start address)
  receive download data from standard input to buffer  ... done  (← indicate input complete size from general output to
(2048 byte)                                                    internal-RAM)
  write download data from buffer to eeprom          ... done  (← write from internal-RAM to EEPROM complete)
  verify download data between eeprom and buffer     ... done  (← verify complete)
```

List 8.12 EEPROM Sample: Download Command Parameter Error

```
I2C EEPROM Writer
> d 0 0                                  (← argument needs two parameters)
  parameter error : (3) - - -            (← indicate the number of parameters for error)
> d 8ff                                  (← start address over device max capacity (2 Kbytes))
  parameter error : (2) d 8ff            (← indicate parameter for error)
```

List 8.13 EEPROM Sample: Help Command Execution

```
I2C EEPROM Writer
> h                                   (← indicate command help)
  r [addr] [size]                     (← read command argument is start address and read size)
  w [addr] [data]                     (← write command argument is start address and 8-bit data)
  d [addr]                            (← download command argument is start address)
  h                                   (← help command)
```

List 8.14 EEPROM Sample: Other Command Execution (command error)

```
I2C EEPROM Writer
> x 0 800                             (← the commands except r, R/w, W/d, D/h, H are error))
  Command error !!
```

9. Development Tool Specific Settings

9.1 Arm

9.1.1 Startup

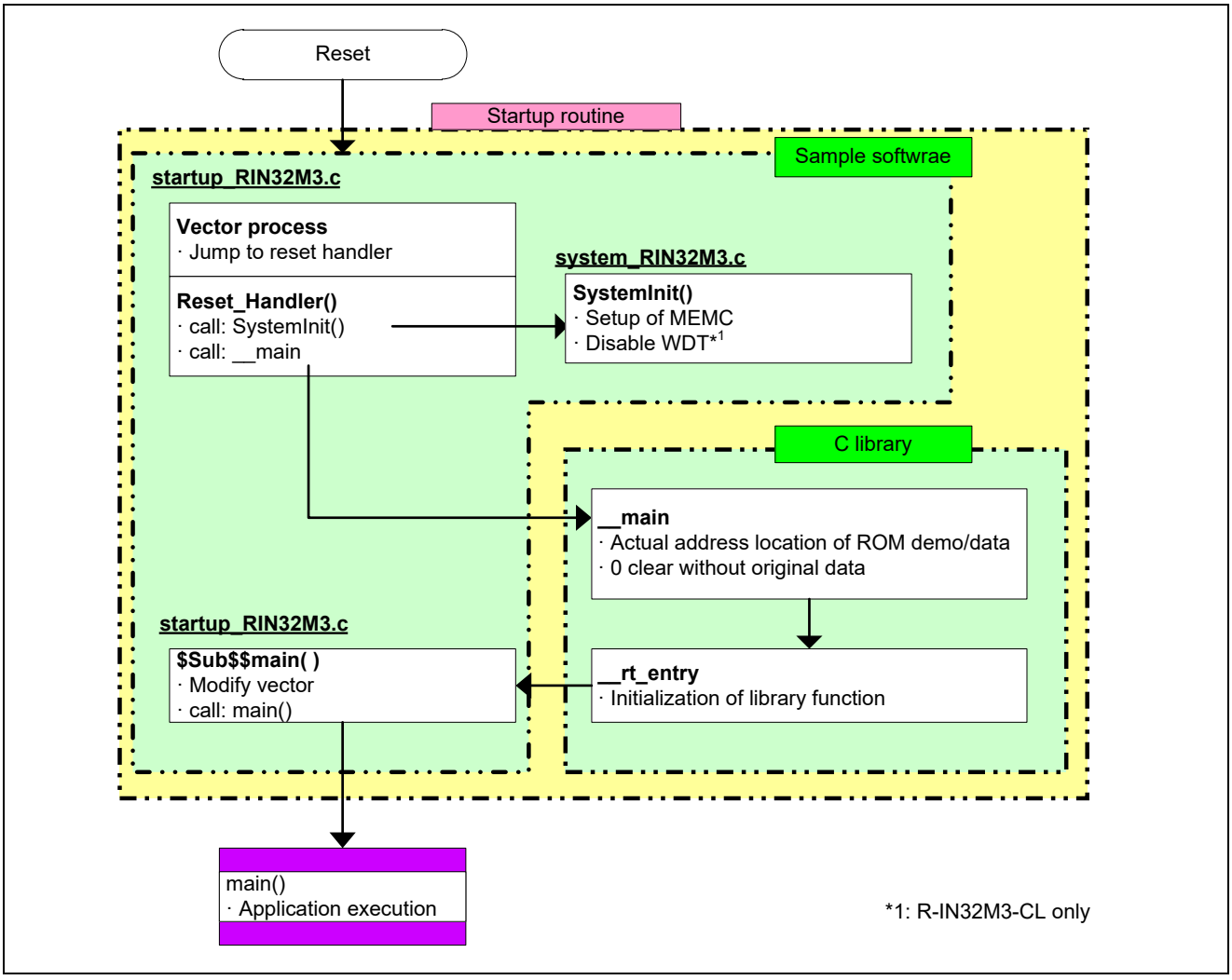

Figure 9.1 Startup Routine with Arm

9.1.2 Replacing Library Functions

Library functions are redefined in syscalls.c.

Table 9.1 Replacing Arm Library Functions

Function Name	Description
fputc	Handles transmission of one byte of data to the UART. The return value is Tx data.
fgetc	Handles reception of one byte of data from the UART and echoes back the result of reception. The return value is Rx data.
ferror	Handles processing for a file error. Processing is not implemented. The return value is EOF.
_sys_exit	Handles processing for exiting the system. The internal function enters an endless loop.
_ttywrch	Handles transmission of one byte of data to the UART. There is no return value.
_clock_init	No operation
clock	The elapsed time (clock_t type) is returned with the interval counter of the interval timer. The time precision is 80 ns.

9.2 GNU

9.2.1 Startup

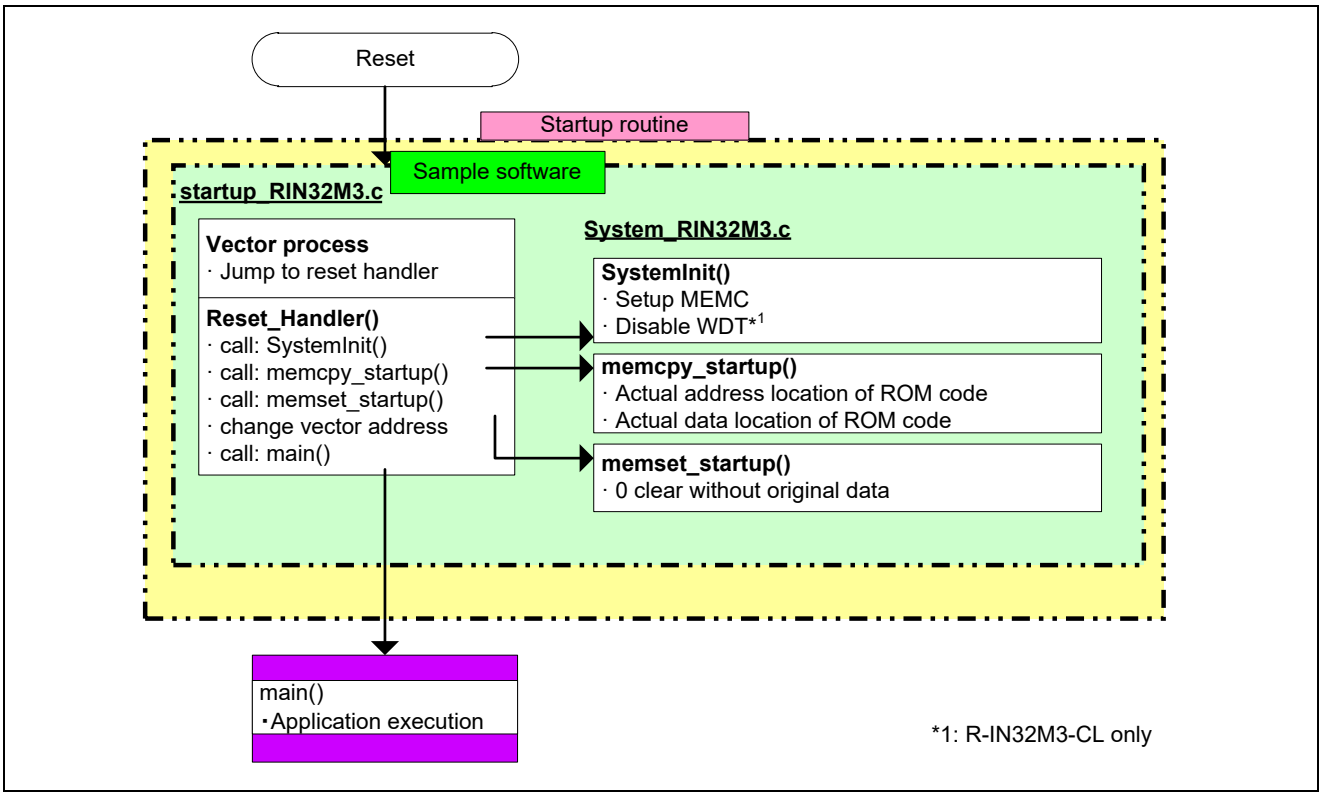

Figure 9.2 Startup Routine with GNU

9.2.2 Replacing Library Functions

Library functions are redefined in syscalls.c.

Table 9.2 Replacing GCC Library Functions

Function	Description
_read_r	Handles reception of 1 byte of data from the UART and echoes back the result of reception. The return value is 1 (the number of received bytes).
_lseek_r	No operation. The return value is 0.
_write_r	Handles transmission of specified bytes of data to the UART. The return value is the number of bytes for transmission.
_open_r	No operation. The return value is -1.
_close_r	No operation. The return value is 0.
_sbrk_r	Checks whether the value of the stack pointer is the same as that of the heap area. The return value: -1 = the area is same If the area is NOT the same, the address where the heap ends is returned.
_fstat_r	Initializes the file structure to 0.The return value is 0.
_isatty_r	No operation. The return value is 1.
_times_r	The user timer is returned (tms.tms_utime) with the interval counter of the interval timer. The elapsed user timer is 80 ns. The return value is -1. 0 is always returned, for system time (tms. tms_stime), sub process user time (tms. tms_cutime) and sub process system time (tms. tms_cutime).

9.3 IAR

9.3.1 Startup

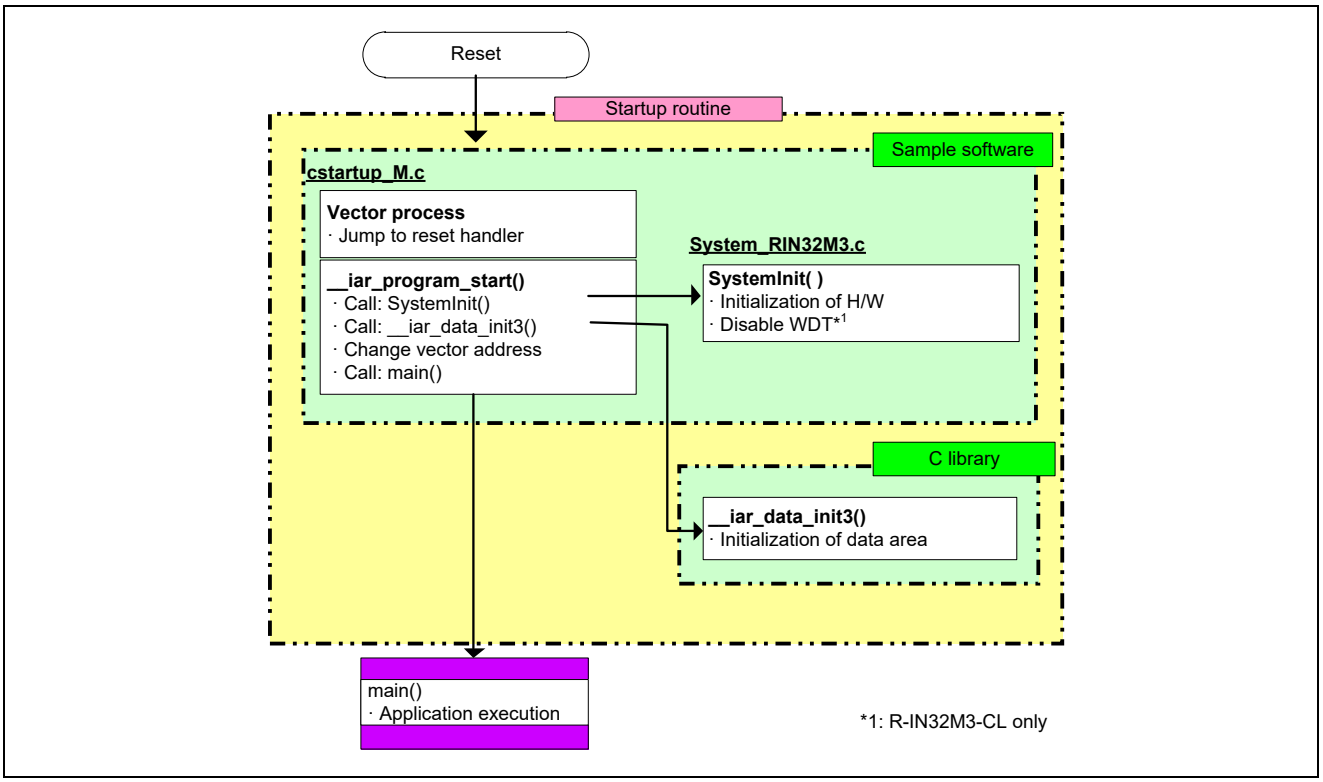

Figure 9.3 Startup Routine with IAR

9.3.2 Replacing Library Functions

Library functions are redefined in syscalls.c.

Table 9.3 Replacing IAR Library Functions

Function Name	Description
__write	Handles transmission of 1 byte of data to the UART. The return value is Tx data size.
__read	Handles reception of 1 byte of data from the UART. The return value is Rx data size.
clock	The elapsed time (clock_t type) is returned with the interval counter of the interval timer. The time precision is 80 ns.

REVISION HISTORY	R-IN32M3 Series Programming Manual: Driver

Rev.	Date	Description	
		Page	Summary
1.00	Mar. 29, 2013	-	First edition issued
2.00	Jun. 13, 2013	1	*Figure 1.1 Layer construction diagram of Sample software* is updated, CC-Link IE and CC-Link are also supported by 3rd party
		2	*Table1.1List of software development tools (Tool chain)* is updated. *Table1.2List of software development tools (development environment)* is updated
		4	*Table 2.2File construction of include file directory* Add file: “RIN32M3_CL.h”
		5	*Table 2.3File construction of library directory* Fix description: “Project file for H/W-RTOS driver” -> “IAR project file”
		21	*Table 5.1Timer driver function list* Fix timer function name
		23	*5.2.1Timer module initialization* Add remark
		24	*5.2.2Initialization of interval timer* Fix omitted letter “timer_”, add remark
		25	*5.2.3Initialization of one count time (triggered by software) Add remark*
		26	*5.2.4Initialization of one count timer (triggered by hardware)* Add remark
		54	*Table 6.2Parallel FlashROM driver function list* Fix word “CFI table” to “CFI data”
		56	*6.2.2EEPROM data write* Fix “(4)Function”,doesn’t wait and doesn’t restart.
		57	*6.2.3EEPROM data read* Fix “(4)Function”,doesn’t wait and doesn’t restart.
		62	*6.3.5CFI data read* Fix word “CFI table” to “CFI data”
		64	*6.4.2Data program to Serial FlashROM* Remove words “(erase and write)”
		69	*7.2EEPROM sample* Add caution about end-of-line character
		-	Add back cover
3.00	Sep. 27, 2013	1	*Figure 1.1 Layer construction diagram of Sample software* Add DMAC. Fix CC-Link and CC-Link IE part.
		2	*Table1.1List of software development tools (Tool chain)* Change IAR version: 6.50 -> 6.60.1
		5	*Table 2.3File construction of library directory* Delete make related files.
		6	*Table 2.4Directory construction of source directory* Delete directory path.
		6	*Table 2.5File construction of drivers’ directory* Delete hwos related files.
		8	*Table 2.7File construction of sample application directory* Delete unused files:Cortex-M3_sample.uer, main.dep, main.dni
		8	*Table 2.7File construction of sample application directory* Add files: scat_boot_sflash.ld, boot_serialflash.icf, init.mac
		8	*Table 2.7File construction of sample application directory* Change file name:“cortex-M3_sample.mvp(mvr)”->”rin32m3ec.mvp(mvr)”, “scat.ld”->”scat_boot_extrom.ld”, “main.icf”->”boot_norflash.icf”
		9	*Table 2.8File construction of startup associated directory* Fix description in syscalls.c.
		9	*Table 2.8File construction of startup associated directory* Add file:vectors_rom.c
		10	*Figure 3.1 File relation diagram* Change the display.
		19	*Table 4.3Constant (systeml)* Change the UART channel number
		20	*Table 4.5Macro define for compile with condition* Change define file.
		75	*8.1.2Replace library functions* Fix section title
		77	*8.2.2Replace library functions* Fix section title

Rev.	Date	Description	
		Page	Summary
3.00	Sep. 27, 2013	78	Figure 9.4 Startup routine with IAR Change structure
		79	*8.3.2Replace library functions* Fix section title
4.00	Dec. 26, 2013	All	Typos are fixed.
		8	*Table 2.7File construction of sample application directory* Delete J-Link setting file.
		20	*Table 4.5Macro define for compile with condition* Add macro “RIN32M3_CL” for R-IN32M3-CL device.
		41-44	Changed the word “macro” to “CSI controller” in CSI section.
5.00	Feb. 28, 2017	2	*Table 1.1 List of Software Development Tools (Tool Chain)* Entries for “ARM” under “Tool Chain” changed to information of “MDK-ARM” (Modification)
		8	*Table 2.7 Configuration of Files in Directories for the Sample Applications* ARM files changed to those for MDK-ARM (Modification)
		9	*Table 2.8 Configuration of Files in Startup Directories* ARM files for RealView changed to those for MDK-ARM (Modification)
		20	*Table 4.5 Macro Definitions for Conditional Compilation* Entry under “Definition File” for OSLESS corrected (Error correction)
		21-22	*5. Definitions of R-IN32M3 Registers (RIN32N3.h)*, added (Complement)
		23	*Table 6.1 Timer Driver Functions* Function timer_onecount_swtrg_init of TAUJ2 deleted (Deletion)
		-	*6.2 Timer Control* Function timer_onecount_swtrg_init of TAUJ2 deleted (Deletion)
		29	*6.2.5 Stopping Timer* Description corrected (Error correction)
		35	*6.4.1 Initialization of IIC Controller* Description of the IIC controller corrected, remark added (Modification)
		69	*Figure 8.1 Flow Chart of OS-less Sample* Key input range corrected (Error correction)
		70	*List 8.2 OS-less Sample: LED Control (key input = ’0’ to ’7’)* Key input range corrected (Error correction)
		77	*9.1.2 Replacing Library Functions* Description added (Description added)
		79	*9.2.2 Replacing Library Functions* Description added (Description added)
		81	*9.3.2 Replacing Library Functions* Description added (Description added)
6.00	Dec. 28, 2018	2	Table 1.1 List of Software Development Tools (Tool Chain) The information on the recommended in-circuit emulator was changed
		11, 12, 15, 16, 17	3.2.1 Memory Maps Note describing that the addresses the instruction RAM mirror area (768 Kbytes) where access actually occurs will change according to the select boot mode, was added. Figure 3.2 Entire Memory Map (R-IN32M3-EC) Figure 3.3Entire Memory Map (R-IN32M3-CL) Figure 3.7 External MCU Interface Space (R-IN32M3-EC) Figure 3.8 External MCU Interface Space (R-IN32M3-CL)

Rev.	Date	Description	
		Page	Summary
6.00	Dec. 28, 2018	11, 12	Figure 3.2 Entire Memory Map (R-IN32M3-EC) Figure 3.3 Entire Memory Map (R-IN32M3-CL) Locations of instruction RAM area and instruction RAM mirror area, corrected
		15, 16, 17	Figure 3.7 External MCU Interface Space (R-IN32M3-EC) Figure 3.8 External MCU Interface Space (R-IN32M3-CL) Instruction RAM area corrected to Instruction RAM mirror area
		25	Table 6.8 CAN Driver Functions The table for the CAN driver functions was added
		36	6.4.1 Initialization of IIC Controller (4) Function Timing setting values were modified and description of PCLK was added as remark 2.
		47	6.5.5 Confirmation of Received Data (for Slave), (4) Function The description was corrected. (Tx mode → Rx mode)
		57 to 76	6.9 CAN Control The section on the CAN control was added
		—	Error corrected, description modified, and contents and expressions adjusted
6.01	Apr. 19, 2019	2	*1.2 Development Environment* , updated.

[Memo]

R-IN32M3 Series Programming Manual: Driver

Publication Date: Rev.1.00 Mar. 29, 2013
Rev.6.01 Apr. 19, 2019

Published by: Renesas Electronics Corporation

R-IN32M3 Series
Programming Manual: Driver

Renesas Electronics Corporation

R18UZ0009EJ0601

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

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

Renesas Electronics Corporation
TOYOSU FORESIA, 3-2-24 Toyosu, Koto-ku, Tokyo 135-0061, Japan

Renesas Electronics America Inc.
1001 Murphy Ranch Road, Milpitas, CA 95035, U.S.A.
Tel: +1-408-432-8888, Fax: +1-408-434-5351

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 101-T01, Floor 1, Building 7, Yard No. 7, 8th Street, Shangdi, Haidian District, Beijing 100085, China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai 200333, China
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

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

Renesas Electronics Malaysia Sdn.Bhd.
Unit No 3A-1 Level 3A Tower 8 UOA Business Park, No 1 Jalan Pengaturcara U1/51A, Seksyen U1, 40150 Shah Alam, Selangor, Malaysia
Tel: +60-3-5022-1288, Fax: +60-3-5022-1290

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL 2nd Stage, Indiranagar, Bangalore 560 038, India
Tel: +91-80-67208700

Renesas Electronics Korea Co., Ltd.
17F, KAMCO Yangjae Tower, 262, Gangnam-daero, Gangnam-gu, Seoul, 06265 Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5338

Colophon 6.0