Programming Manual

R-IN32M4-CL3

Programming Manual: Driver

Document number: R18UZ0076EJ0100
Issue date : Nov. 1, 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 communications ASSP (Application Specific Standard Product) "R-IN32M4-CL3" (R9J03G019GBG) 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. Last four digits of document number (described as ****) indicate version information of each document. Please download the latest document from our web site and refer to it.

Documents related to R-IN32M4-CL3

Document Name	Document Number
R-IN32M4-CL3 User's Manual: Hardware	R18UZ0073EJ****
R-IN32M4-CL3 User's Manual: Gigabit Ethernet PHY	R18UZ0075EJ****
R-IN32M4-CL3 User's Manual: Board	R18UZ0074EJ****
R-IN32M4-CL3 User's Manual: CC-Link IE TSN	R18UZ0070EJ****
R-IN32M4-CL3 User's Manual: CC-Link IE Field	R18UZ0071EJ****
R-IN32M4-CL3 Programming Manual: Driver	This Manual
R-IN32M4-CL3 Programming Manual: OS	R18UZ0072EJ****

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

1. Outline

Sample software code showing examples of the usage of each peripheral module of the R-IN32M4-CL3 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-IN32M4-CL3.

1.1 Structure

The layered structure of sample software is shown below.

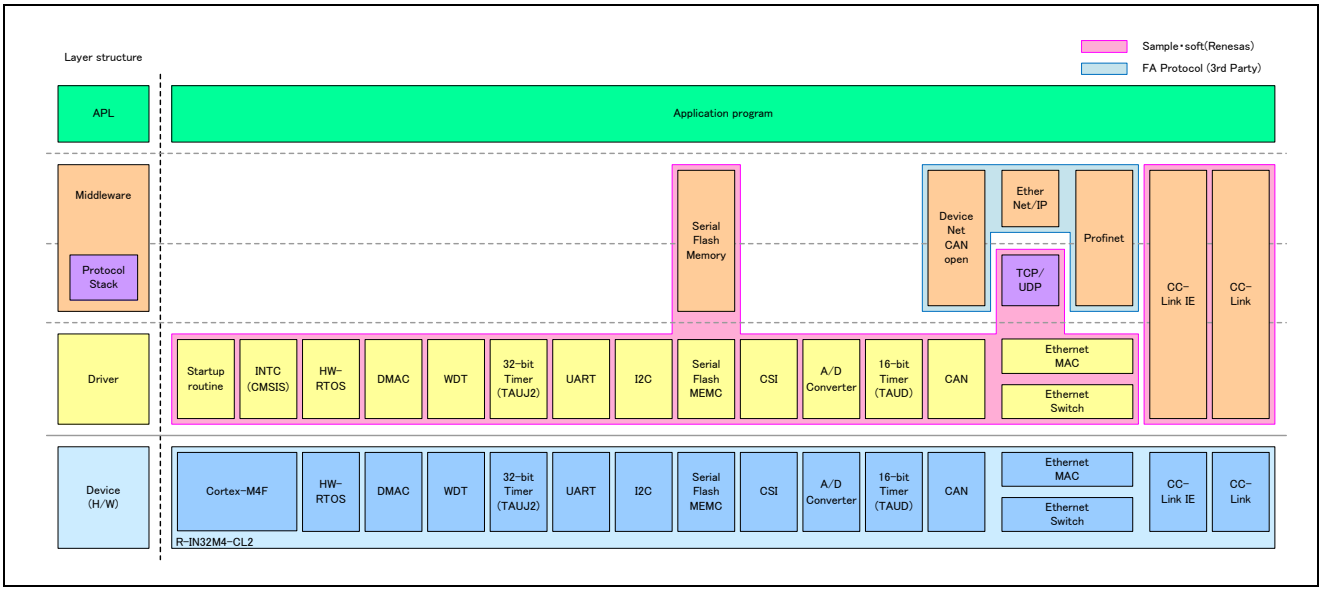

Figure 1.1 Layered Structure of the Sample Software

1.2 Development Environment

The explanation of software development tools is given below.
This sample software adopts Arm® Cortex® Microcontroller Software Interface Standard (CMSIS) V4.5.0.
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 V8.42.1 ~ Latest Version (Please use the latest version) (IAR Systems)			I-jet JTAGjet-Trace-CM (IAR Systems)

2. Configuration of Files

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

2.1 IAR Embedded Workbench for Arm

2.1.1 Configuration of Directories

Table 2.1 Configuration of Directories for IAR Embedded Workbench for Arm

Directory	Contents
./	Directory of sample software
./CMSIS	Directory of Cortex-M4-related definition: equal to CMSIS provided by Arm
./Device	Directory of device specific files
./Device/Renesas/RIN32M4	Directory of R-IN32M4-CL3 specific programs
./Device/Renesas/RIN32M4/Include	Directory of include files
./Device/Renesas/RIN32M4/Library	Directory of library files
./Device/Renesas/RIN32M4/Source	Directory of source files

2.1.2 ./Device/Renesas/RIN32M4/Include: Include Files

The configuration of include files is listed below.

Table 2.2 Configuration of Files in Include File Directories

Directory	File	Contents
./can/	can.h	Prototype declaration of CAN driver
./csi/	csi.h	Prototype declaration of CSI driver
./dmac/	dmac.h	Prototype declaration of DMA driver
./ether/	ether_phy_init.h	Prototype declaration of GbE-PHY driver
./iic/	iic.h	Prototype declaration of I2C driver
./sromc/	sromc.h	Prototype declaration of serial flash ROM driver
./taud/	taud.h	Prototype declaration of 16-bit timer (TAUD) driver
./timer/	timer.h	Prototype declaration of 32-bit timer (TAUJ2) driver
./uart/	uart.h	Prototype declaration of UART driver
./wdt/	wdt.h	Prototype declaration of WDT driver
./	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)
	RIN32M4_CL3.h	Device definition file for R-IN32M4-CL3
	system_RIN32M4.h	System information definition based on CMSIS

2.1.3 ./Device/Renesas/RIN32M4/Library: Library Files

The configuration of library files is listed below.

Table 2.3 Configuration of Files in Library Directories

Directory	File	Contents
./IAR/	libos.a	H/W-RTOS driver library
	libEtherPHY.a	GbE-PHY driver library

2.1.4 ./Device/Renesas/RIN32M4/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

2.1.4.1 ./Device/Renesas/RIN32M4/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
./can/	can_cfg.c	CAN configuration table
./can/	can_data.c	CAN data driver
./can/	can_init.c	CAN initialization driver
./can/	can_mode.c	CAN mode driver
./can/	can_status.c	CAN status driver
./can/	can_xfer.c	CAN transfer driver
./csi/	csi.c	CSI driver
./dmac/	dmac.c	DMAC driver
./iic/	iic.c	I2C driver
./sromc/	sromc.c	Serial flash ROM Driver
./taud/	taud.c	16-bit timer (TAUD) driver
./timer/	timer.c	32-bit timer (TAUJ2) driver
./uart/	uart.c	UART driver
./wdt/	wdt.c	WDT driver

2.1.4.2 ./Device/Renesas/RIN32M4/Source/Middleware: Middleware Files

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

Table 2.6 Configuration of Files in Middleware Directories

Directory	File	Contents
./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

2.1.4.3 ./Device/Renesas/RIN32M4/Source/Project: Sample Application

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

Table 2.7 Configuration of Directories for the Sample Software Applications

Directory	Contents
./TS-R-IN32M4-CL3	Sample application directory for Tessera Technologies evaluation board
./TS-R-IN32M4-CL3/board_init.c	Board-dependent setting source file
./TS-R-IN32M4-CL3/board_init.h	Board-dependent setting header file
./TS-R-IN32M4-CL3/osless_sample	OS-less sample application

Table 2.8 Configuration of Files in Directories for the OS-Less Sample Application

Directory	File	Contents
./osless_sample/	main.c	Main processing source 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)
	boot_serialflash.icf	Linker scripts (boot code: serial flash ROM allocated)
	iram.icf	Linker scripts (boot code: instruction RAM allocated)
	init.mac	Debugger macro script file

2.1.4.4 ./Device/Renesas/RIN32M4/Source/Templates: Startup Files

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

Table 2.9 Configuration of Files in Startup Directories

Directory	File	Contents
./Templates/IAR/	cstartup_M.c	Startup file (for IAR)
	vectors_M.c	Vector definition file
	vectors_rom.c	Vector definition file (for ROM boot)
	syscalls.c	To replace library functions (for IAR)
./Templates/	system_RIN32M4.c	Startup file (common)

3. Software Development Procedure

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

3.1 Design Flow

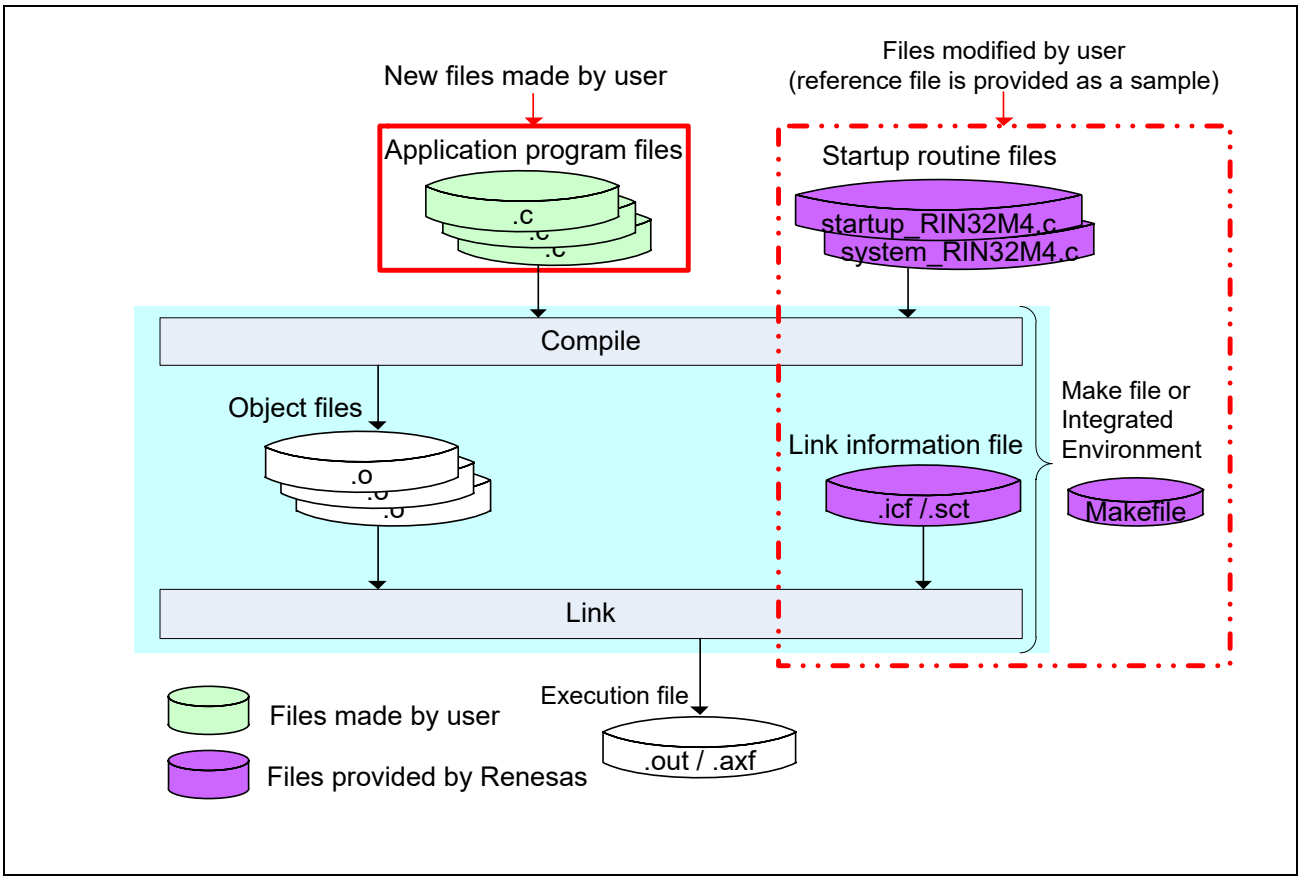

Figure 3.1 File Correlation Diagram

3.2 Memory Maps

3.2.1 Memory Maps

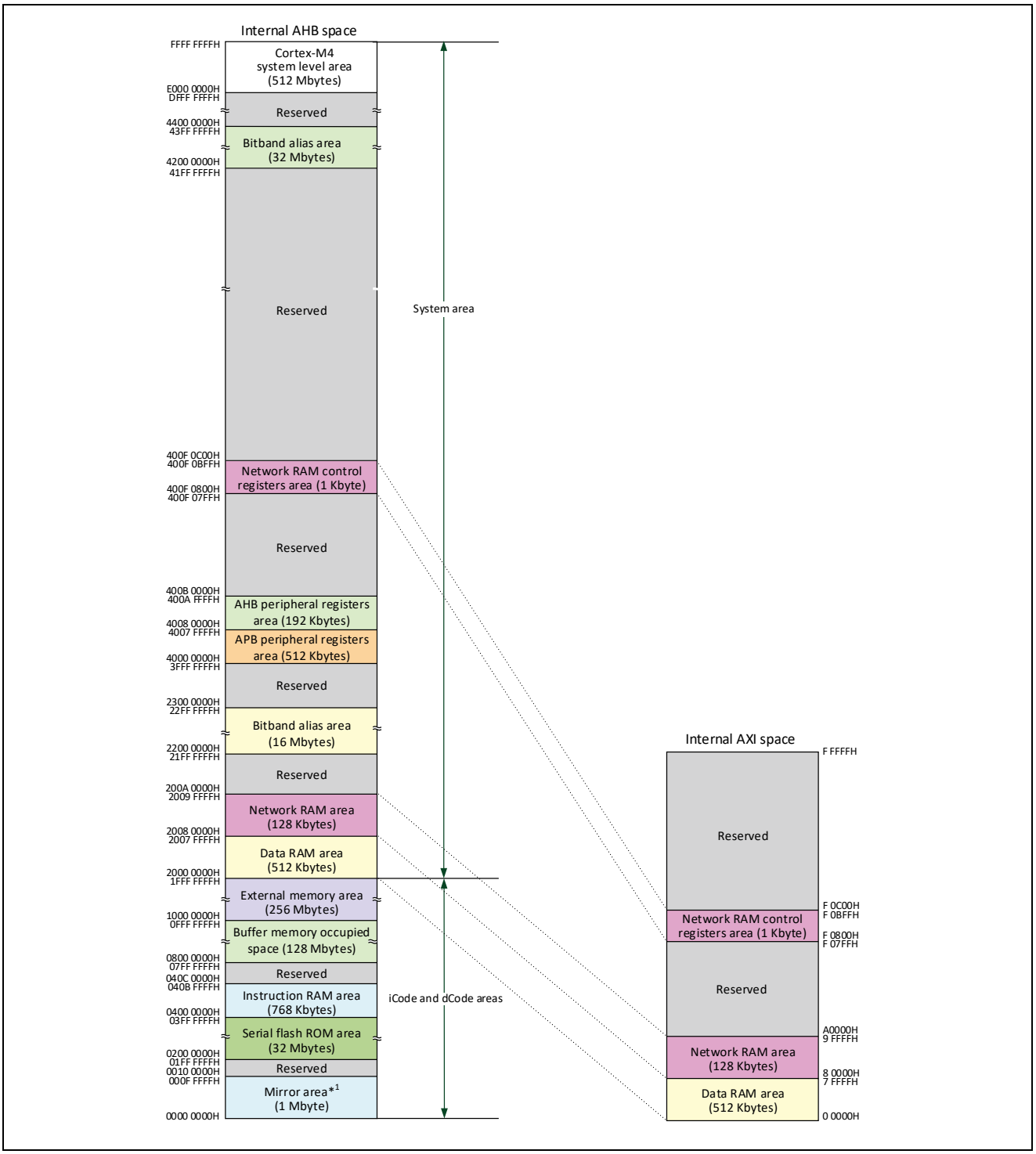

Figure 3.2 Memory Map (All)

Note 1. The mirror area is an area where one of the external memory area/serial flash memory area/instruction RAM area can be mirrored by setting the external pins (BOOT0, BOOT1). For details, refer to Section 7 "Booting Procedure" in the R-IN32M4-CL3 User's Manual: Hardware.

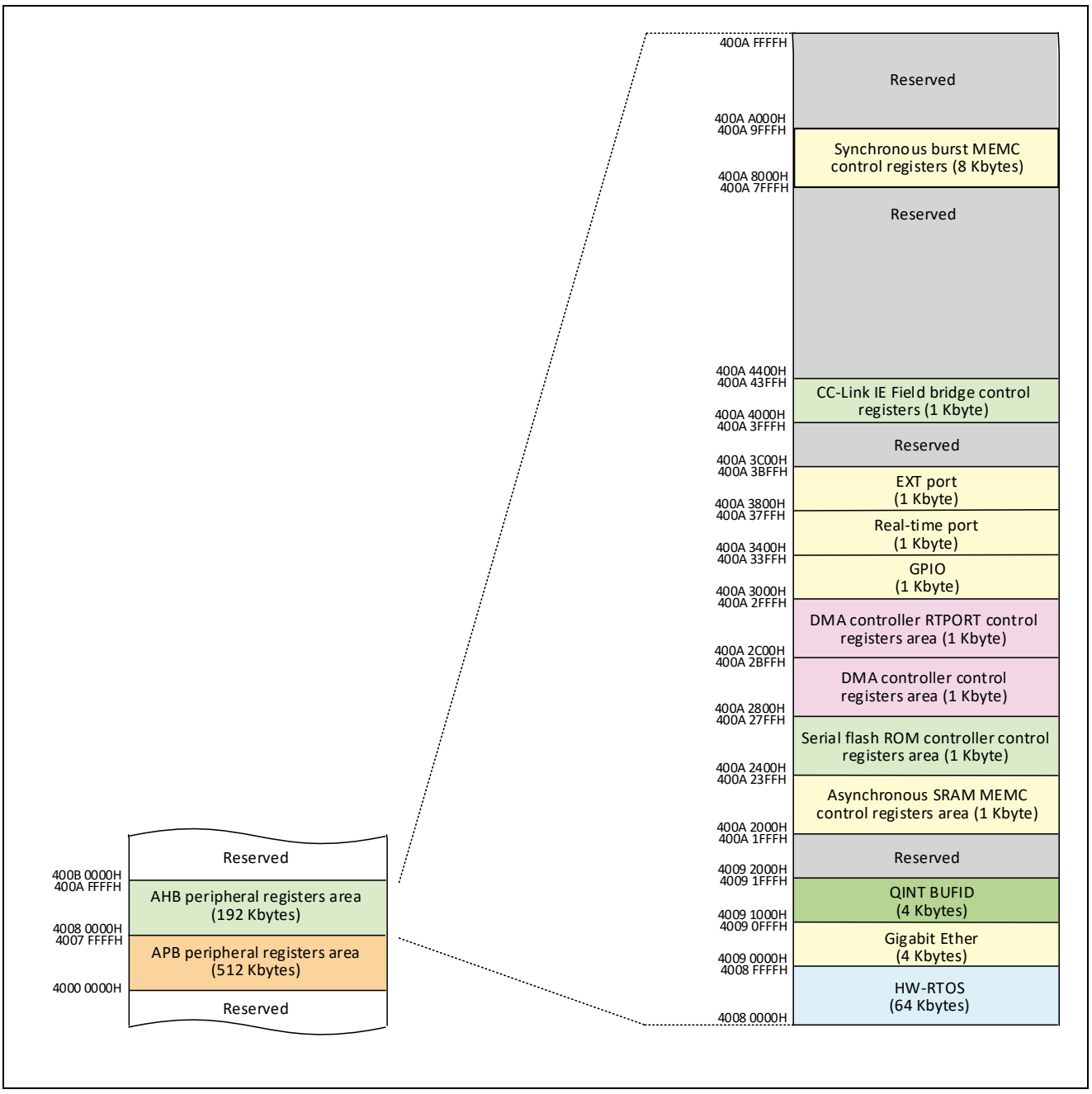

Figure 3.3 Memory Map (AHB Peripheral Registers Area)

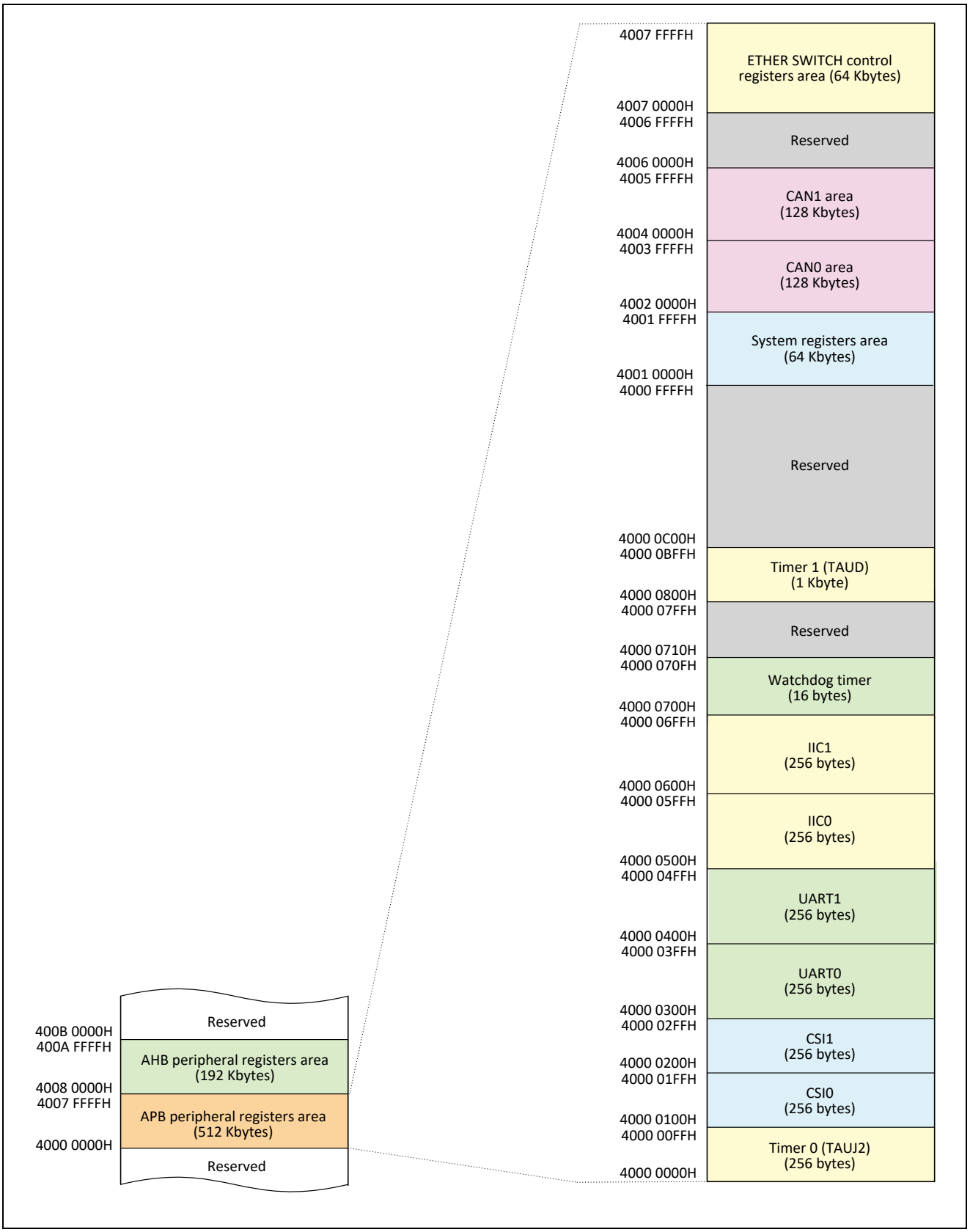

Figure 3.4 Memory Map (APB Peripheral Registers Area)

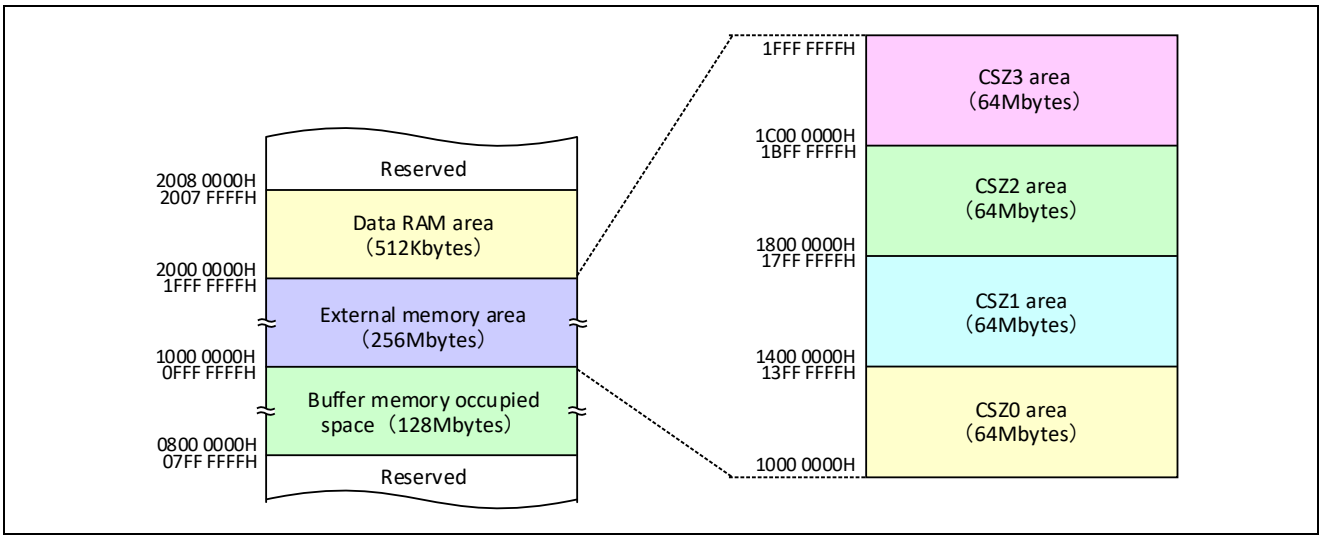

Figure 3.5 Memory Map (External Memory Area 23mm package)

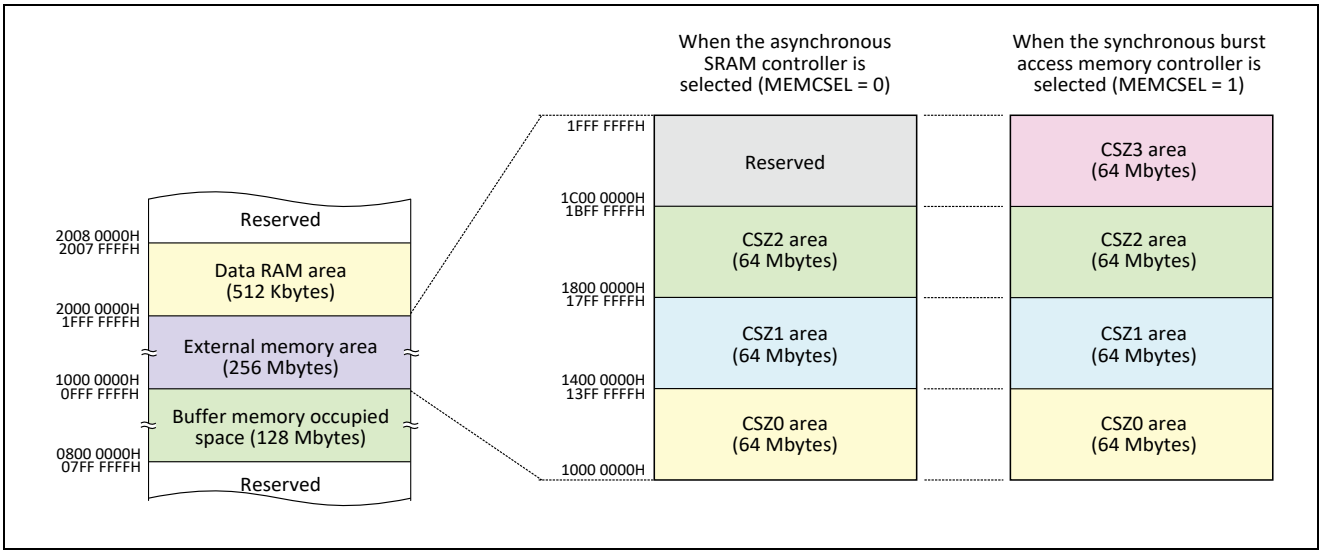

Figure 3.6 Memory Map (External Memory Area 17mm package)

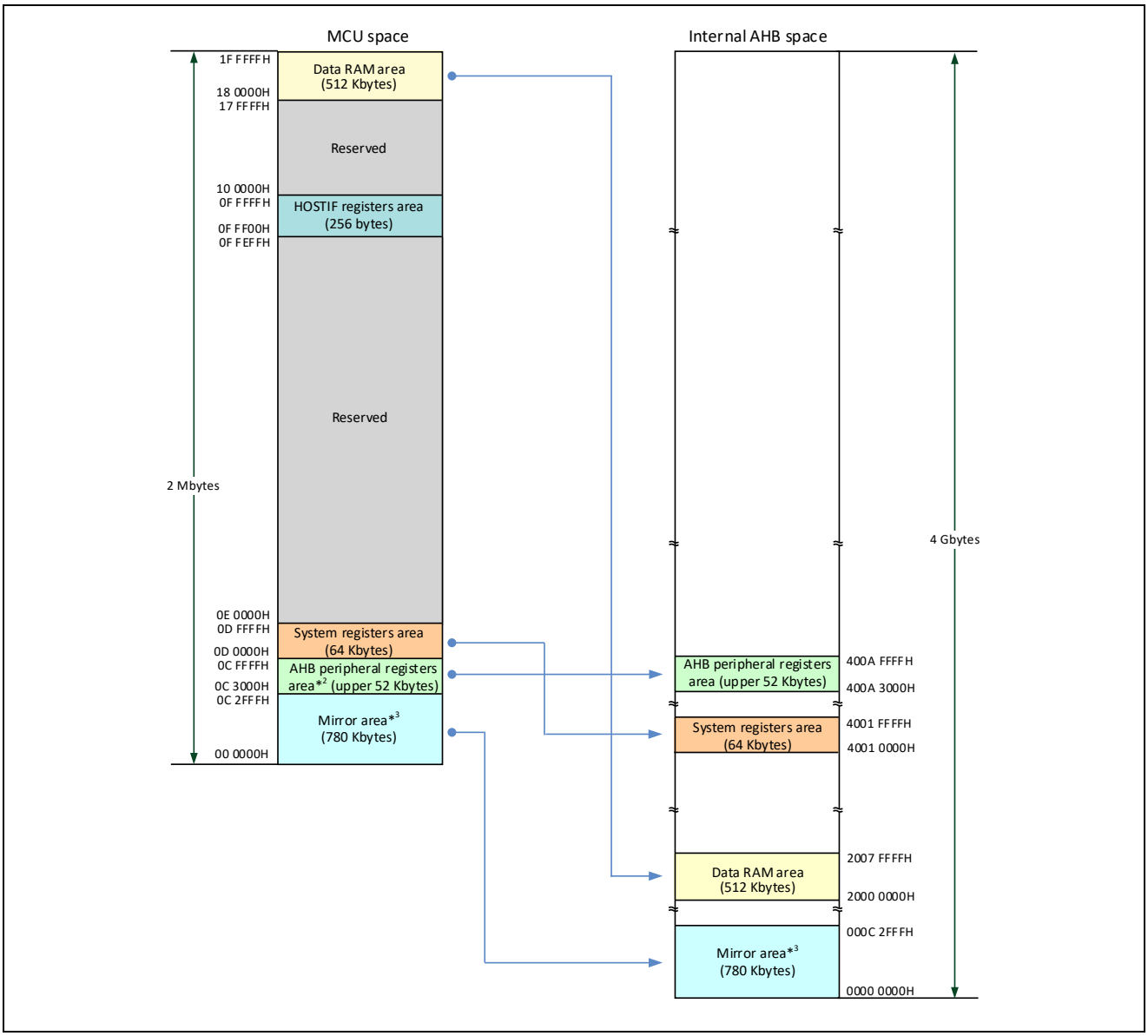

Figure 3.7 External MCU Interface Space

Note 1. The upper 52 Kbytes of the AHB peripheral registers area cover the range from the GPIO area to the synchronous burst memory controller control registers. For details, refer to "Figure 3.3 Memory Map (AHB Peripheral Registers Area)".

2. The mirror area is an area where one of the external memory area/serial flash memory area/instruction RAM area can be mirrored by setting the external pins (BOOT0, BOOT1).
When access is from the external MCU interface, the following areas can be accessed.
• When BOOT0, BOOT1 = 01 (External serial flash ROM boot): Reserved area (access prohibited)
• When BOOT0, BOOT1 = 10 (External MCU boot): Instruction RAM area
For the lists of slaves accessible by the bus masters, refer to Section 6 "Bus Configuration" in the R-IN32M4-CL3 User's Manual: Hardware.
For details on boot mode, refer to Section 7 "Booting Procedure" in the R-IN32M4-CL3 User's Manual: Hardware.

3.2.2 Program Allocation Example

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

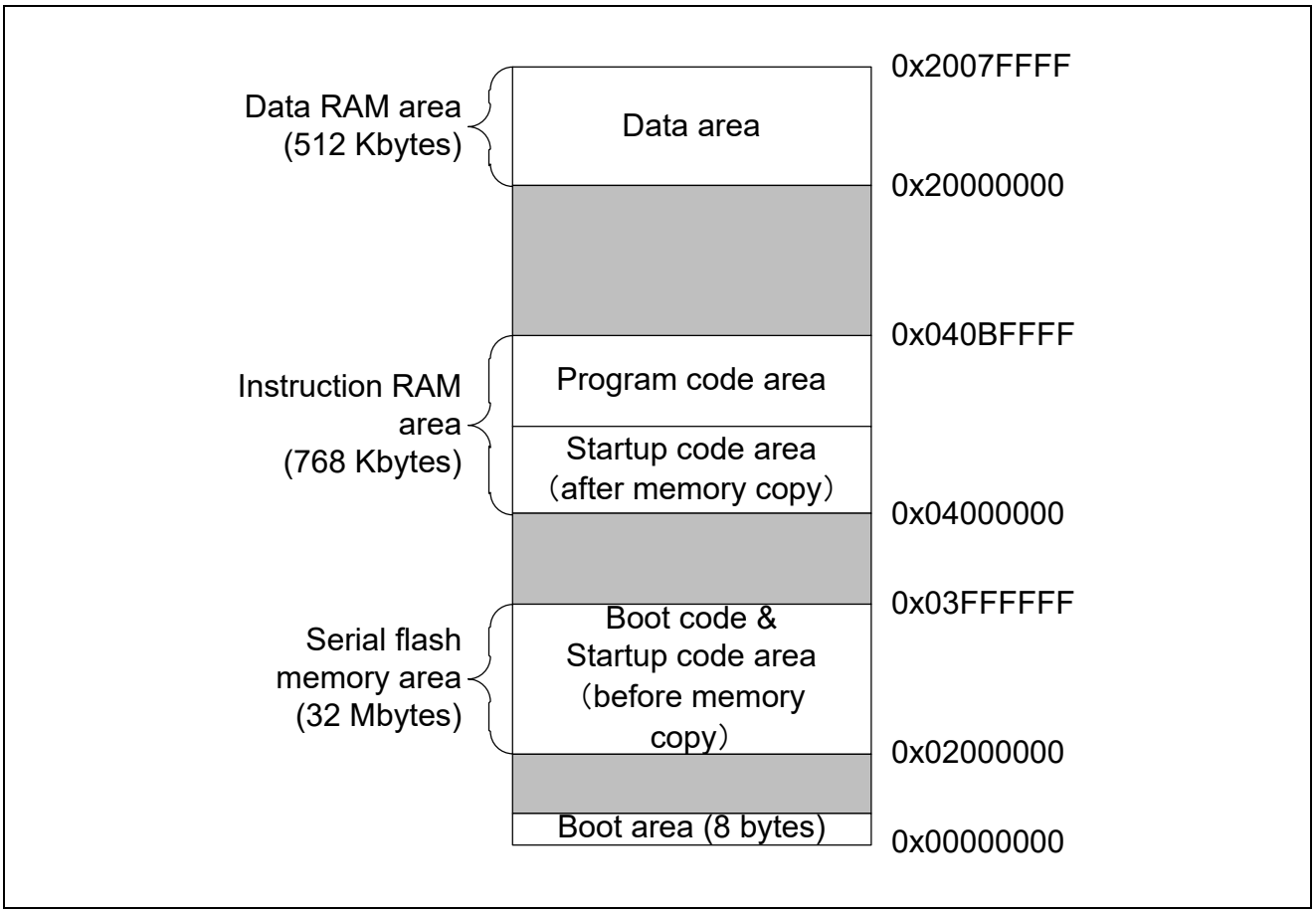

Figure 3.8 Program Allocation Example

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
RIN32M4_SYSCLK	100000000	Clock frequency of the system (unit: Hz)
SYS_UART_CH	1	UART channel number in the system

Table 4.4 Constants (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 Conditional Compilation Definition

A macro definition for conditional compilation is as follows.

Table 4.5 Macro Definition for Conditional Compilation

Definition	Description	File
OSLESS	Determines whether HW-RTOS is used or not. If defined: HW-RTOS is not used. If not defined: HW-RTOS is used.	cstartup_M.c

5. R-IN32M4-CL3 Register Definition

The include file RIN32M4_CL3.h defines interrupts and registers of R-IN32M4-CL3. Table 5.1 and Table 5.2 list register definitions. For details of the individual registers, refer to the relevant section of the document under "Reference of Register Details" in each table.

5.1 APB Peripheral Registers

Table 5.1 APB Peripheral Register Definition

#define	Function	Reference of Register Details
#define RIN_TMR_BASE	32-bit timer register (TAUJ2)	"R-IN32M4-CL3 User's Manual: Hardware" section 19
#define RIN_CSI0_BASE	CSI channel-0 register	"R-IN32M4-CL3 User's Manual: Hardware" section 23
#define RIN_CSI1_BASE	CSI channel-1 register	
#define RIN_UART0_BASE	UART channel-0 register	"R-IN32M4-CL3 User's Manual: Hardware" section 22
#define RIN_UART1_BASE	UART channel-1 register	
#define RIN_IIC0_BASE	I2C channel-0 register	"R-IN32M4-CL3 User's Manual: Hardware" section 24
#define RIN_IIC1_BASE	I2C channel-1 register	
#define RIN_WDT_BASE	Watchdog timer register	"R-IN32M4-CL3 User's Manual: Hardware" section 21
#define RIN_TAUD_BASE	16-bit timer register (TAUD)	"R-IN32M4-CL3 User's Manual: Hardware" section 20
#define RIN_SYS_BASE	System register	"R-IN32M4-CL3 User's Manual: Hardware " each section
#define RIN_CAN0_BASE	CAN channel-0 register	"R-IN32M4-CL3 User's Manual: Hardware" section 25
#define RIN_CAN1_BASE	CAN channel-1 register	
#define RIN_ETHSW_BASE	Ethernet switch register	"R-IN32M4-CL3 User's Manual: Hardware" section 13

5.2 AHB Peripheral Registers

Table 5.2 AHB Peripheral Register Definition

#define	Function	Reference of Register Details
#define RIN_HWOS_BASE	HW-RTOS register	"R-IN32M4-CL3 User's Manual: Hardware" section 10
#define RIN_ETH_BASE	Gigabit Ethernet MAC register	"R-IN32M4-CL3 User's Manual: Hardware" section 12
#define RIN_MEMC_BASE	Asynchronous SRAM MEMC register	"R-IN32M4-CL3 User's Manual: Hardware" section 14
#define RIN_SROM_BASE	Serial Flash ROM MEMC register	"R-IN32M4-CL3 User's Manual: Hardware" section 17
#define RIN_DMAC0_BASE	DMAC channel-0 control register	"R-IN32M4-CL3 User's Manual: Hardware" section 18
#define RIN_DMAC1_BASE	DMAC channel-1 control register	
#define RIN_DMAC2_BASE	DMAC channel-2 control register	
#define RIN_DMAC3_BASE	DMAC channel-3 control register	
#define RIN_DMAC0_LINK_BASE	DMAC channel-0 link register	
#define RIN_DMAC1_LINK_BASE	DMAC channel-1 link register	
#define RIN_DMAC2_LINK_BASE	DMAC channel-2 link register	
#define RIN_DMAC3_LINK_BASE	DMAC channel-3 link register	
#define RIN_DMAC_CTRL_BASE	DMA control register	
#define RIN_RTDMAC_BASE	RTDMAC control register	
#define RIN_RTDMAC_LINK_BASE	RTDMAC link register	
#define RIN_RTDMAC_CTRL_BASE	RTDMA control register	
#define RIN_GPIO_BASE	Port register	"R-IN32M4-CL3 User's Manual: Hardware" section 27
#define RIN_RTPORT_BASE	RT port register	
#define RIN_EXTPORT_BASE	EXT port register	
#define RIN_CCI_BRG_BASE	CC-Link IE Field bridge control register	"R-IN32M4-CL3 User's Manual: Hardware" section 26
#define RIN_SMC_BASE	Synchronous burst access MEMC register	"R-IN32M4-CL3 User's Manual: Hardware" section 15

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 32-Bit Timer (TAUJ2) Driver Functions

Function	Description
clock_init	Initialization of TAUJ2 system timer (channel-0)
timer_interval_init	Initialization of TAUJ2 interval timer
timer_onecount_hwtrg_init	Initialization of TAUJ2 one-count timer (hardware trigger)
timer_start	Start TAUJ2 timer
timer_stop	Stop TAUJ2 timer
timer_check_act	Check TAUJ2 timer activation

Table 6.2 UART Driver Functions

Function	Description
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	Description
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	Description
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 Functions

Function	Description
dmac_memcpy	Copy memory (DMA transfer)

Table 6.6 Serial Flash ROM Driver Functions

Function	Description
sromc_init	Initialization of SPI bus controller (normal reading)
sromc_dual_init	Initialization of SPI bus controller (Fast Read Dual I/O)
sromc_quad_init	Initialization of SPI bus controller (Fast Read Quad I/O)
sromc_write	Write data to SPI bus
sromc_read	Read data form SPI bus

Table 6.7 WDT Driver Functions

Function	Description
wdt_init	Initialization of watchdog timer
wdt_interrupt_init	Initialization of watchdog timer (enable 75% interrupt)
wdt_start	Watchdog timer activation
wdt_clear	Counter clearing
wdt_wait_reset	Wait for reset

Table 6.8 16-Bit Timer (TAUD) Driver Functions

Function	Description
taud_interval_init	Initialization of TAUD interval timer
taud_onecount_hwtrg_init	Initialization of TAUD one-count timer (hardware trigger)
taud_start	Start TAUD timer
taud_stop	Stop TAUD timer
taud_check_act	Check TAUD timer activation

Table 6.9 CAN Driver Functions

Function	Description
can_enable	Enabling CAN controller
can_init	Initialization of CAN Controller
can_shutdown	Forced Termination of CAN Controller
can_get_mode	Acquisition of CAN Operating Mode
can_set_mode	Setting CAN Operating Mode
can_get_id_data_dlc	Acquisition of CAN Reception Data (CANID, Data, DLC)
can_get_data_dlc	Acquisition of CAN Reception Data (Data, DLC)
can_set_id_data_dlc	Setting CAN Transmission Data (CAN_ID, Data, DLC)
can_set_data	Setting CAN Transmission Data
can_tx_req	Request of CAN Data Transmission
can_get_txinfo	Acquisition of CAN Data Transmission Information
can_get_rxinfo	Acquisition of Reception Buffer Number of CAN Data
can_get_ch_status	Acquisition of CAN Cannel Status
can_clr_ch_status	Clearing CAN Channel Status
can_get_bus_staus	Acquisition of CAN Bus Status

Table 6.10 GbE-PHY Driver Function

Function	Description
ether_phy_init	Initialization of GbE-PHY

6.2 32-Bit Timer (TAUJ2) Control

6.2.1 Initialization of TAUJ2 System Timer (Channel-0)

clock_init

(1) Description

Initialization of the TAUJ2 system timer (channel-0)

(2) C-Language Format

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

(3) Parameter

None

(4) Function

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

(5) Return Value

None

Remark: This API uses "TAUJ2TTINm input position detection" described in "TAUJ2 Operations" in "R-IN32M4-CL3 User's Manual: Hardware".

6.2.2 Initialization of TAUJ2 Interval Timer

timer_interval_init

(1) Description

Initialization of the TAUJ2 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	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
  - > Counter 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 is NOT 1 to 42,949 ms

Remark: This API uses "interval timer" described in "TAUJ2 Operations" in "R-IN32M4-CL3 User's Manual: Hardware".

6.2.3 Initialization of TAUJ2 One-Count Timer (Hardware Trigger)

timer_onecount_hwtrg_init

(1) Description

Initialization of the TAUJ2 one-count timer (hardware trigger)

(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 (1 to 42,949,672 us)
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 timing for counting once is NOT 1 to 42,949,672 us

Remark: This API uses "delay counting" described in "TAUJ2 Operations" in "R-IN32M4-CL3 User's Manual: Hardware".

6.2.4 Starting TAUJ2 Timer

timer_start

(1) Description

Starting the TAUJ2 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 TAUJ2 Timer

timer_stop

(1) Description

Stopping the TAUJ2 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 TAUJ2 Timer Activation

timer_check_act

(1) Description

Checking the TAUJ2 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 operating 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_RIN32M4.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_RIN32M4.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_RIN32M4.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_RIN32M4.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

This function initializes IIC 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, see the R-IN32M4-CL3 User's Manual: Hardware.
2. PCLK cycle = 10 ns

(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

Remark 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, place the controller 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, place the controller 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 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, change the setting as below.
  - > Stopping Tx operation
  - > Enabling Rx operation
- If the transfer mode selection argument is Tx mode, change the setting as below.
  - > Enabling Tx operation
  - > Disabling Rx operation

(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 (for Normal Reading)

sromc_init

(1) Description

Initialization of the SPI bus controller (for normal reading)

(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 : normal read
- 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 Initialization of SPI Bus Controller (for Fast Read Dual I/O)

sromc_dual_init

(1) Description

Initialization of the SPI bus controller (for Fast Read Dual I/O)

(2) C-Language Format

```
void sromc_dual_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 : 50 MHz
  - > Serial clock default level : high
- Serial flash ROM read mode setting : fast read dual I/O
- 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 cycle
    - Device select signal of the SPI bus : 1.5 serial clock cycles
    - Serial data output : 0.5 serial clock cycles
  - > Hold cycle
    - Device select signal of the SPI bus : 1.5 serial clock cycles
    - Serial data output : 0.5 serial clock cycles

(5) Return Value

None

6.7.3 Initialization of SPI Bus Controller (for Fast Read Quad I/O)

sromc_quad_init

(1) Description

Initialization of the SPI bus controller (for Fast Read Quad I/O)

(2) C-Language Format

```
void sromc_quad_init( void );
```

(3) Parameter

None

(4) Function

This function initializes the serial flash ROM controller.
Using fast read quad I/O requires setting the serial flash ROM controller. The settings are contained in the middleware. For details, refer to section 7.3.3, Initialization of Serial Flash ROM Controller (for Fast Read Quad I/O)".

- Serial flash ROM clock setting
  - > Serial flash ROM clock frequency : 50 MHz
  - > Serial clock default level : high
- Serial flash ROM read mode setting : fast read quad I/O
- 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 cycle
    - Device select signal of the SPI bus : 1.5 serial clock cycles
    - Serial data output : 0.5 serial clock cycles
  - > Hold cycle
    - Device select signal of the SPI bus : 1.5 serial clock cycles
    - Serial data output : 0.5 serial clock cycles

(5) Return Value

None

6.7.4 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

6.7.5 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 Initialization of Watchdog Timer (Enabling 75% Interrupts)

wdt_interrupt_init

(1) Description

Initialization of the watchdog timer (enabling 75% interrupts)

(2) C-Language Format

```
ER_RET wdt_interrupt_init( void );
```

(3) Parameter

None

(4) Function

This function initializes the watchdog timer. Set WDTA0MD.WDTA0WIE to 1 to enable 75% interrupt requests. This leads to the generation of an interrupt before the counter overflows.
When an interrupt occurs, the driver clears the counter.

- Error mode : Reset mode (reset when the counter overflows)
- 75% interrupt request : Enabled
- Counter overflow interval time : 1.342 s
- Window open period : 100%

(5) Return Value

Return Value	Meaning
ER_OK	Initialization succeeded

6.8.3 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	Start operation succeeded

6.8.4 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 clearing succeeded

6.8.5 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 16-Bit Timer (TAUD) Control

6.9.1 Initialization of TAUD Interval Timer

taud_interval_init

(1) Description

Initialization of the TAUD interval timer

(2) C-Language Format

```
ER_RET taud_interval_init( uint8_t ch, uint16_t i_time );
```

(3) Parameter

I/O	Parameter	Description
I	uint8_t ch	Channel selection argument 0: channel-0 1: channel-1 2: channel-2 … 15: channel-15
I	uint16_t i_time	Interval (1 to 655 us)

(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 15 or the interval argument is other than 1 to 655.

- Timer clock specification
  > Counter 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 15 - The interval is NOT 1 to 655 us

Remark: This API uses "interval timer" described in "TAUD Operations" in "R-IN32M4-CL3 User's Manual: Hardware".

6.9.2 Initialization of TAUD One-Count Timer (Hardware Trigger)

taud_onecount_hwtrg_init

(1) Description

Initialization of the one-count timer of TAUD (hardware trigger)

(2) C-Language Format

```
ER_RET taud_onecount_hwtrg_init( uint8_t ch,uint16_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 … 15: channel-15
I	uint16_t	o_time	Time for counting once (1 to 655 us)
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.
The timer counter clock has to be 100 MHz in this mode.

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 success
ER_PARAM	Parameter error - The selected channel is NOT 0 to 15 - The time for counting once is NOT 1 to 655 us

Remark: This API uses "delay counting" described in "TAUJ2 Operations" in "R-IN32M4-CL3 User's Manual: Hardware".

6.9.3 Starting TAUD Timer

taud_start

(1) Description

Starting the TAUD timer

(2) C-Language Format

```
ER_RET taud_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 … 15: channel-15

(4) Function

This function starts the timer selected by the channel selection argument.
A parameter error is returned if the selected channel is other than 0 to 15.

(5) Return Value

Return Value	Meaning
ER_OK	Timer started
ER_PARAM	Parameter error - The selected channel is NOT 0 to 15

6.9.4 Stopping TAUD Timer

taud_stop

(1) Description

Stopping the TAUD timer

(2) C-Language Format

```
ER_RET taud_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 … 15: channel-15

(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 15

6.9.5 Checking TAUD Timer Activation

taud_check_act

(1) Description

Checking the TAUD timer activation

(2) C-Language Format

```
ER_RET taud_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 … 15: channel-15

(4) Function

This function checks whether the timer selected by the channel selection argument is operating 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 15

6.10 CAN Control

6.10.1 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

6.10.2 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

6.10.2.2 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 25.13, Baud Rate Settings, in the R-IN32M3 Series User's Manual: Hardware

(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 25.7.4, Mask Function, in the R-IN32M3 Series User's Manual: Hardware

(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

6.10.3 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

6.10.4 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

6.10.5 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

6.10.6 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

6.10.7 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

6.10.8 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

6.10.9 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

6.10.10 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

6.10.11 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

6.10.12 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

6.10.13 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

6.10.14 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

6.10.15 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

6.11 GbE-PHY Control

6.11.1 Initialization of GbE-PHY

ether_phy_init

(1) Description

Initialization of GbE-PHY

(2) C-Language Format

```
void ether_phy_init( void );
```

(3) Parameter

None

(4) Function

This function initializes GbE-PHY.
If the initialization failed, ER_NG is returned as a return value.

After GbE-PHY is reset, be sure to call this initialization function to make initial settings for GbE-PHY. For details, see section 9, Usage Note.

(5) Return Value

None

Caution: This function includes processing for releasing and re-setting protection by the SYSPCMD register within the function.
Accordingly, if protection has been released by the SYSPCMD register before calling this function, protection is set with this function.

7. Middleware

Functions of the middleware are explained in this section.
This serial flash ROM driver is designed for use with R-IN32M4-CL3 evaluation board from Tessera Technologies and serial flash ROM "MX25L6433F".

7.1 Lists of Middleware Functions

The API functions in this sample software are listed below.

Table 7.1 Parallel Flash ROM Driver Functions

Function	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 7.2 Serial Flash ROM Driver Functions

Function	Description
sflash_init	Initialization of serial flash ROM control (for normal reading)
sflash_dual_init	Initialization of serial flash ROM control (for Fast Read Dual I/O)
sflash_quad_init	Initialization of serial flash ROM control (for Fast Read Quad I/O)
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 Parallel Flash ROM Control

7.2.1 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

7.2.2 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 sum of the addr argument and the size argument is over the maximum size of parallel flash ROM

7.2.3 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 sum of the addr argument and the size argument is over the maximum size of parallel flash ROM

7.2.4 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 sum of the addr argument and the size argument is over the maximum size of parallel flash ROM

7.2.5 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	Argument of read address designation

(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

7.3 Serial Flash ROM Control

7.3.1 Initialization of Serial Flash ROM Controller (for Normal Reading)

sflash_init

(1) Description

Initialization of the serial flash ROM controller (for normal reading)

(2) C-Language Format

```
ER_RET sflash_init( void );
```

(3) Parameter

None

(4) Function

This function initializes the serial flash ROM controller. Call the sromc_init function described in section 6.7.1, Initialization of SPI Bus Controller (for Normal Reading) to initialize the SPI bus controller, and then disable quad mode of the serial flash ROM controller.
If setting of the serial flash ROM controller failed, ER_NG is returned.

(5) Return value

Return Value	Meaning
ER_NG	Initialization failure - Setting of serial flash ROM controller failed

7.3.2 Initialization of Serial Flash ROM Controller (for Fast Read Dual I/O)

sflash_dual_init

(1) Description

Initialization of the serial flash ROM controller (for Fast Read Dual I/O)

(2) C-Language Format

```
ER_RET sflash_dual_init( void );
```

(3) Parameter

None

(4) Function

This function initializes the serial flash ROM controller. Call the sromc_init function described in section 6.7.2, Initialization of SPI Bus Controller (for Fast Read Dual I/O) to initialize the SPI bus controller, and then disable quad mode of the serial flash ROM controller.
If setting of the serial flash ROM controller failed, ER_NG is returned.

(5) Return value

Return Value	Meaning
ER_NG	Initialization failure - Setting of serial flash ROM controller failed

7.3.3 Initialization of Serial Flash ROM Controller (for Fast Read Quad I/O)

sflash_quad_init

(1) Description

Initialization of the serial flash ROM controller (for Fast Read Quad I/O)

(2) C-Language Format

```
ER_RET sflash_quad_init( void );
```

(3) Parameter

None

(4) Function

This function initializes the serial flash ROM controller. Call the sromc_quad_init function described in section 6.7.3, Initialization of SPI Bus Controller (for Fast Read Quad I/O) to initialize the SPI bus controller, and then enable quad mode of the serial flash ROM controller.
If setting of the serial flash ROM controller failed, ER_NG is returned.

(5) Return Value

Return Value	Meaning
ER_NG	Initialization failure - Setting of serial flash ROM controller failed

7.3.4 Programming Data to Serial Flash ROM

sflash_program

(1) Description

Programming data to serial flash ROM

(2) C-Language Format

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

(3) Parameter

I/O	Parameter		Description
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.
If the sum of the addr argument and the size argument is over the maximum size of serial flash ROM, ER_PARAM is returned.

(5) Return Value

Return Value	Meaning
ER_OK	Success
ER_PARAM	Parameter error - The sum of addr argument and size argument is over the maximum size of serial flash ROM.

7.3.5 Reading Data from Serial Flash ROM

sflash_read

(1) Description

Reading data from serial flash ROM

(2) C-Language Format

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

(3) Parameter

I/O	Parameter		Description
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 to the buffer.
If the sum of the addr argument and the size argument is over the maximum size of serial flash ROM, ER_PARAM is returned.

(5) Return Value

Return Value	Meaning
ER_OK	Success
ER_PARAM	Parameter error - The sum of addr argument and size argument is over the maximum size of serial flash ROM.

7.3.6 Erasing Serial Flash ROM Data

sflash_erase

(1) Description

Erasing serial flash ROM data

(2) C-Language Format

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

(3) Parameter

I/O	Parameter	Description
I	uint32_t addr	Address where data 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.
If the sum of the addr argument and the size argument is over the maximum size of serial flash ROM, ER_PARAM is returned.

(5) Return Value

Return Value	Meaning
ER_OK	Success
ER_PARAM	Parameter error: - The sum of addr argument and size argument is over the maximum size of serial flash ROM.

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 real-time OS.

8.1.1 Flow of OS-Less Sample

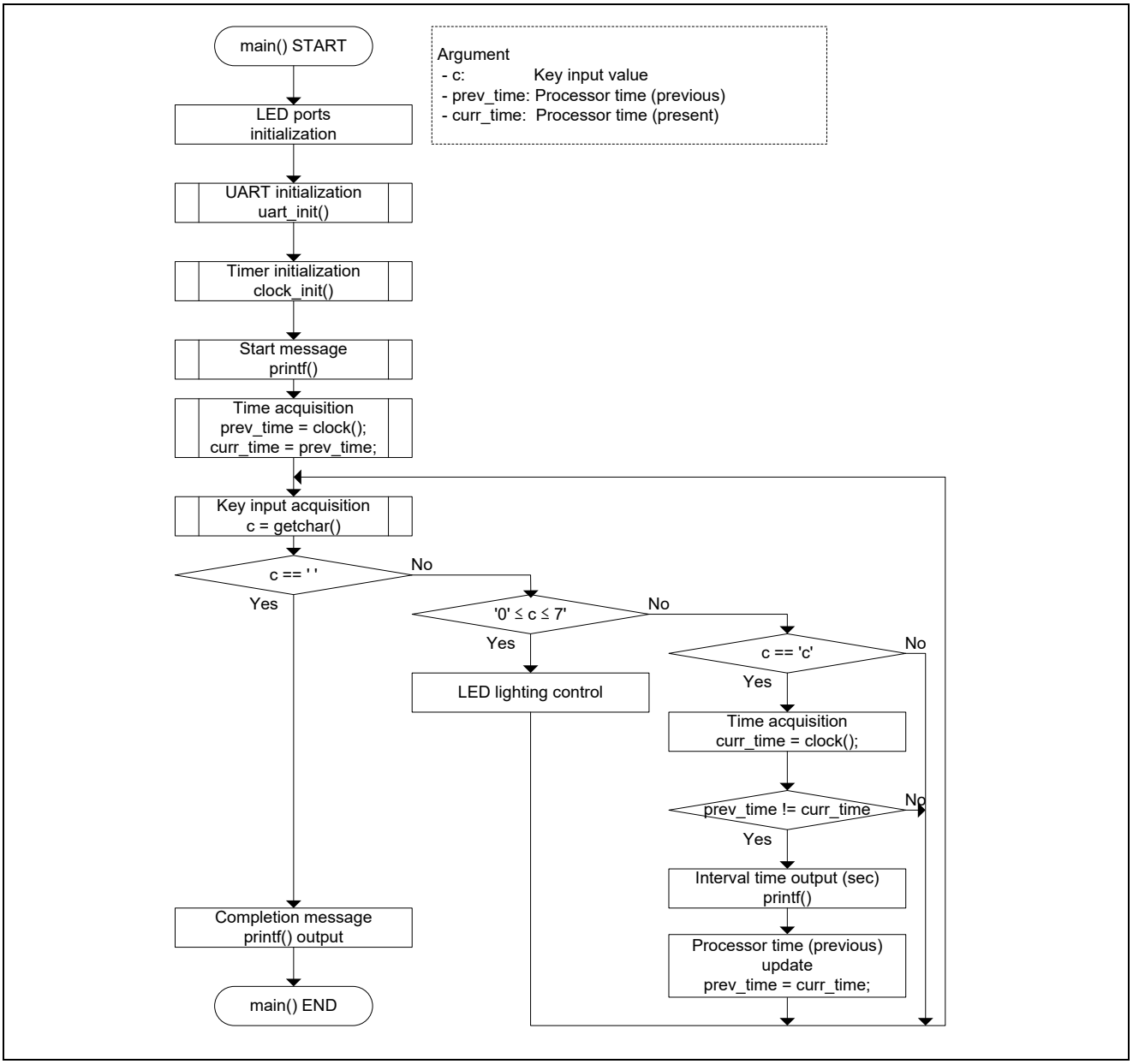

Figure 8.1 Flow Chart of OS-Less Sample

8.1.2 Result of Execution

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

(1) OS-less Sample: Starting

```
hello world
- compiler  =IAR ANSI C/C++ Compiler V7.40.7.9843/W32 for ARM
- boot mode =Serial Flash
```

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

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

```
hello world
- compiler  =IAR ANSI C/C++ Compiler V7.40.7.9843/W32 for
ARM
- boot mode =Serial Flash                          (← The level of LED0 is inverted 1 time)
0                                                  (← The level of LED1 is inverted 2 times)
11                                                 (← The level of LED2 is inverted 3 times)
222                                                (← The level of LED3 is inverted 4 times)
3333                                               (← The level of LED4 is inverted 5 times)
44444                                              (← The level of LED5 is inverted 6 times)
555555                                             (← The level of LED6 is inverted 7 times)
6666666                                            (← The level of LED7 is inverted 8 times)
77777777
```

(3) OS-less Sample: Indication of Interval Time (key input = ’c’)

```
hello world
- compiler  =IAR ANSI C/C++ Compiler V7.40.7.9843/W32 for
ARM
- boot mode =Serial Flash                          (← indicate time from program start to 1st 'C' key input)
c (interval =464369 ms)                            (← Indicate time from 1st 'C' key input to 2nd 'C' key input)
c (interval =2771 ms)                              (← indicate time from 2nd 'C' key input to 3rd 'C' key input)
c (interval =187253 ms)
```

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

```
hello world
- compiler  =IAR ANSI C/C++ Compiler V7.40.7.9843/W32 for
ARM
- boot mode =Serial Flash                          (← break from main processing by ' ' input)

bye
```

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

```
hello world
- compiler  =IAR ANSI C/C++ Compiler V7.40.7.9843/W32 for
ARM
- boot mode =Serial Flash                                    (← not taken any step)
890-^\!"#$%&'()=~|qwertyuiop@[QWERTYUIO`{                    (← not taken any step)
asdfghjkl;:]ASDFGHJKL+*}zxvbnm,./\ZXVBNM<>?
```

9. Usage Note

When the power is turned on or the GbE-PHY is reset (except a software reset inside the GbE-PHY), make sure to initialize the GbE-PHY (see section 6.11.1, Initialization of GbE-PHY).
In the initialization of GbE-PHY, settings related to characteristics and performance of the GbE-PHY are made.

REVISION HISTORY	R-IN32M4-CL3 Programming Manual: Driver

Rev.	Date	Description	
		Page	Summary
1.00	Nov. 01, 2019	-	First edition issued

[Memo]

R-IN32M4-CL3 Programming Manual: Driver

Publication Date: Rev.1.00 Nov. 01, 2019

Published by: Renesas Electronics Corporation

R-IN32M4-CL3 Programming Manual: Driver

Renesas Electronics Corporation